2012

Parallel Algorithms for Bayesian Networks Structure Learning with Applications in Systems Biology

Olga Nikolova
Iowa State University

Parallel algorithms for Bayesian networks structure learning

with applications in systems biology

by

Olga Nikolova

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Bioinformatics and Computational Biology

Program of Study Committee:
Srinivas Aluru, Co-major Professor
Patrick Schnable, Co-major Professor
Dan Nettleton
Julie Dickerson
Guang Song

Iowa State University

Ames, Iowa

2012

DEDICATION

I dedicate this work to my parents, Svetlana Nikolova and Hristo Nikolov.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Algorithms

ABSTRACT

The expression levels of thousands to tens of thousands of genes in a living cell are controlled by internal and external cues which act in a combinatorial manner that can be modeled as a network. High-throughput technologies, such as DNA-microarrays and next generation sequencing, allow for the measurement of gene expression levels on a whole-genome scale. In recent years, a wealth of microarray data probing gene expression under various biological conditions has been accumulated in public repositories, which facilitates uncovering the underlying transcriptional networks (gene networks). Due to the high data dimensionality and inherent complexity of gene interactions, this task inevitably requires automated computational approaches.

Various models have been proposed for learning gene networks, with Bayesian networks (BNs) showing promise for the task. However, BN structure learning is an NP-hard problem and both exact and heuristic methods are computationally intensive with limited ability to produce large networks. To address these issues, we developed a set of parallel algorithms. First, we present a communication efficient parallel algorithm for exact BN structure learning, which is work-optimal provided that $2^n > p \cdot \log p$, where n is the total number of variables, and p is the number of processors. This algorithm has space complexity within 1.41 of the optimal. Our empirical results demonstrate near perfect scaling on up to $2,048$ processors. We further extend this work to the case of bounded node in-degree, where a limit d on the

number of parents per variable is imposed. We characterize the algorithm's run-time behavior as a function of d, establishing the range $\left[\frac{1}{3}n - \log mn, \lceil \frac{n}{2} \rceil\right)$ of values for d where it affects performance. Consequently, two plateaus regions are identified: for $d < \frac{1}{3}n - \log mn$, where the run-time complexity remains the same as for $d = 1$, and for $d \geq \lceil \frac{n}{2} \rceil$, where the run-time complexity remains the same as for $d = n - 1$. Finally, we present a parallel heuristic approach for large-scale BN learning. This approach aims to combine the precision of exact learning with the scalability of heuristic methods. Our empirical results demonstrate good scaling on various high performance platforms. The quality of the learned networks for both exact and heuristic methods are evaluated using synthetically generated expression data. The biological relevance of the networks learned by the exact algorithm is assessed by applying it to the carotenoid biosynthesis pathway in *Arabidopsis thaliana*.

CHAPTER 1. Introduction

1.1 Background and Motivation

1.1.1 Gene Regulatory Networks

The expression levels of thousands to tens of thousands of genes in a living cell are controlled by internal and external cues which act in a combinatorial manner (with multiple regulatory inputs) in the form of a network. In response to signals, transcription factor proteins (TFs) adjust the transcription rates of target genes which they regulate to ensure that the appropriate amounts of RNA are transcribed and that the necessary proteins are produced by the cell at all times. In many cases TFs' activity levels can be closely approximated by their expression levels. In the cases when TFs' activity levels are determined by other biochemical events, they can often be derived from the expression levels of indirect regulators, e.g. signaling proteins, which in turn control TFs' activity [Pe'er et al., 2001, Segal et al., 2003]. Therefore, when gene expression levels are modeled as random variables, it is expected that interactions between regulators and target genes result in statistical dependencies [Pe'er et al., 2006]. While transcription regulation is a much more complex process involving various epigenetic processes (methylation, acetylation, histone modification, etc.), in this work we term *gene network* the schematic representation of gene interactions as defined by their relative expression levels changes. In reality, genes do not directly interact with other genes. Instead, gene expression

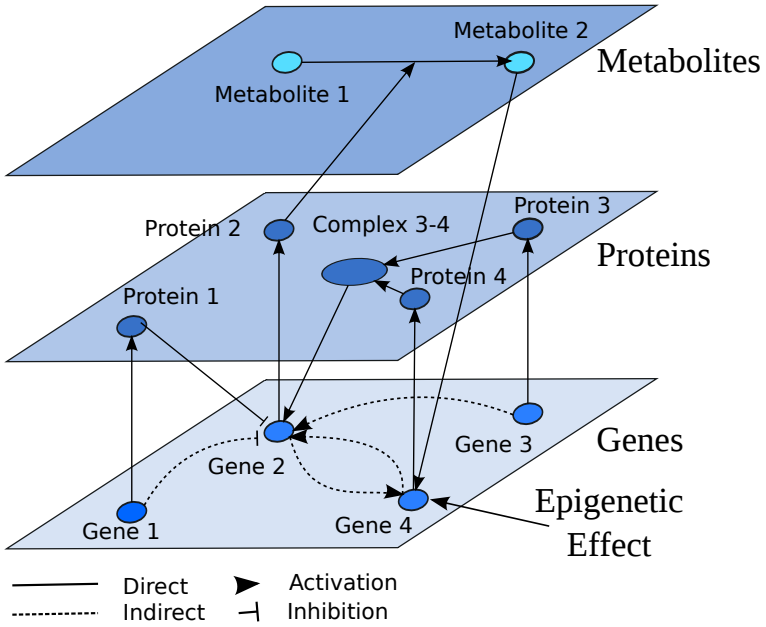


Figure 1.1 Schematic representation of a gene regulatory network. Solid arrows indicate direct interactions. The dashed lines indicate indirect associations between genes that result from projecting the interactions between elements from the three different spaces onto the gene space. Arrows represent activation and bars represent inhibition. (Drawn after Volkhard Helms, Principles of Computational Cell Biology, 2008.)

levels are regulated by the actions of gene products, proteins and metabolites (see Figure 1.1). What is observed in terms of gene expression changes are rather the indirect interactions, which could be visualized as the projection of gene interactions with proteins and metabolites onto the gene space. Gene networks are commonly represented as directed or undirected graphs, where nodes represent genes and edges - interactions (see level A of Figure 1.1).

High-throughput technologies, such as DNA-microarrays and Next Generation Sequencing Technologies, allow for the measurement of gene expression levels on a whole-genome scale [Kotoulas and Siebes, 1999, Schena et al., 1995]. In the recent years a wealth of microarray expression data has been accumulated in public repositories, such as, NASCArrays [Craigon et al., 2004] and ArrayExpress [Brazma et al., 2003], resulting from both small- and large-

scale studies, probing gene expression under various biological conditions. Uncovering the underlying gene networks from observed microarray gene expression data (also referred to as *reverse engineering gene networks*) is an important and difficult problem. Due to the high data dimensionality and inherent complexity of gene interactions, this task inevitably requires automated computational approaches. Various models have been proposed for reverse engineering gene networks including correlation-based methods [Rice et al., 2005], information theory-based methods [Basso et al., 2005], dynamical models [Fuente et al., 2002], as well as, Bayesian networks [Friedman, 2004]. In their community-wide effort for assessing the quality of computational techniques for network reconstruction, Marbach *et al.* concluded that this problem remains open and very challenging [Marbach et al., 2010]. Their findings reveal that more than $1/3^{rd}$ of the methods perform as good as random guessing, and that the majority of current inference methods fail to correctly predict multiple regulatory inputs (combinatorial regulation).

1.1.2 Bayesian Networks as a Model for Gene Regulatory Networks

Bayesian networks (BNs) are graphical models which represent probabilistic relationships between interacting variables in a given domain. In BNs, relationships between variables are depicted in the manner of conditional independencies (presence or absence of edges) and quantitatively assessed by conditional probability distributions [Grzegorczyk and Husmeier, 2008]. BNs provide a holistic view of the interactions within the domain and can be learned from data. BNs have been successfully applied in a variety of different research areas, including decision support systems, language processing, medicine, and bioinformatics [Basilio et al., 2000, Friedman, 2004, Heckerman et al., 1995b]. Certain properties make them especially attractive for gene network induction. One of them is the ability to depict combinatorial

interactions, highlighted as one of the most challenging tasks by [Marbach et al., 2010]. In the search for a globally optimal BN, all possible subsets of variables are evaluated as potential regulators for each gene, in the context of a complete structure, unlike co-expression networks, where individual pair-wise or partial correlations are considered. Another one, is the ability to learn causal edges in presence of perturbation data, which in the context of a biological system can be achieved for example via a gene knockout or over-expression of a given gene [Pe'er et al., 2001]. Further, the scoring functions used in BN structure learning approaches are robust to overfitting. Most scoring functions for BN structure learning employ a penalty term to balance the goodness of fit with the complexity of the given model. Examples of such scoring functions are the information theoretic Bayesian Information Criterion (BIC) [Schwarz, 1978] and Akaike Information Criterion (AIC) [Akaike, 1974], the information theoretic scoring function based on the Minimum Description Length principle (MDL) [Lam and Bacchus, 1994], and Bayesian scores, which estimate the posterior probability of the considered model given the data, such as Bayesian Dirichlet (BD) [Cooper and Herskovits, 1992]) and its extension Bayesian Dirichlet equivalence (BDe) [Heckerman et al., 1995a]. Major difficulties in inferring BNs from data are due to its computational and space complexity. In the following subsections we outline the state-of-the-art approaches and the remaining open problems for both exact and heuristic BN structure learning.

1.1.3 Exact Bayesian Network Structure Learning

In order to learn an optimal BN from data, every possible network configuration over the variables in the given domain is scored, given the observed data. The search space of BN structure learning for n variables corresponds to the space of all possible directed acyclic graphs (DAGs) on the n variables, which is shown to be super-exponential in n [Robinson, 1973]. In

an effort to reduce the search space, marginalization over node orderings was proposed, which successfully reduces the search space to exponential [Koivisto, 2006, Koivisto et al., 2004, Ott et al., Silander and Myllymaki, 2006]. Adopting a canonical representation of the DAGs in conjunction with a decomposable scoring function (where the score of the network can be computed as a sum of score contributions of each variable and its parents) allows for a dynamic programming (DP) approach, described in detail in Chapter 2. However, even with these improvements, and after further limiting the number of possible regulators for each gene, the BN structure learning problem is shown to be NP-hard [Chickering et al., 1994].

State-of-the-art sequential algorithms exploit the improvements described above [Ott et al., Silander and Myllymaki, 2006] but are still unable to scale to domains of even 30 variables. The reason for this is not only in the run time, but also in the memory requirements, which grow exponentially in n as well. In addition, with the large amounts of publicly available microarray data, large samples (thousands of microarray chips) can be compiled after appropriate statistical pre-processing as shown in Aluru *et al.* [Aluru et al., Under Review], and used for network induction. Computational complexity of the utilized scoring functions increases linearly in the number of observations (as the data matrix is scanned), which introduces additional computational cost. Thus, both the computational and memory requirements create a bottleneck for BN structure learning, leaving the problem of computing optimal BNs for both large number of variables and large number of observations open. Even-though heuristic-based algorithms for BN structure learning exist, they trade off optimality for the ability to learn larger networks, and structure learning without additional assumptions remains valuable. Ott *et al.* report that inferring a BN for 20 variables under 173 conditions took approximately 50 hours.

1.1.4 Heuristic Bayesian Network Structure Learning

Due to the exponential complexity of exact learning, exact approaches do not scale to large networks, prompting the development of heuristic approaches [Chen et al., 2006, Cheng, 2002, Chickering, 2002, Friedman et al., Heckerman et al., 1995a, Moore and Wong, 2003, Pe'er et al., 2006, Spirtes et al., 2001, Spirtes and Meek, 1995, Tsamardinos et al., 2006, Wang et al., 2007]. In principle the heuristic approaches can be categorized as either i) score optimization methods, ii) constraint-based methods, or iii) hybrid approaches, which include elements of the first two. A commonly used approach from the first category is a greedy search. It starts with an empty network (all variables are independent), a fully-connected network, or a random network, and proceed to score every possible change, which can be implemented by adding an edge, removing an edge, or reversing an edge of this structure, utilizing a Bayesian scoring criteria. The newly generated structure which results from making the highest-scoring change is accepted and the algorithm repeats this step in a greedy search manner until a structure is selected which optimizes the score [Chickering, 2002, Heckerman et al., 1995a, Moore and Wong, 2003]. In the second category, direct and entailed conditional independencies are learned from the observed data using statistical or information theoretic measures and a graphical criterion called *d-separation*. A partially oriented network is returned which satisfies these constraints [Elidan and Friedman, 2005, Spirtes et al., 2001]. These methods are shown to do better in discovering the sets of parents and children of each variable (undirected network), than identifying the parents of each node (the final DAG). Methods in the third category utilize constraint-based approaches to learn the sets of parents and children for each variable, and then proceed with heuristic score-optimization to learn the final network [Chen et al., 2006, Spirtes and Meek, 1995, Tsamardinos et al., 2006, Wang et al., 2007]. A more in depth literature review is given

in Chapter 4. An extensive evaluation of the performance of the state-of-the-art algorithms, including the *Sparse Candidate* [Friedman et al.], *Three Phase Dependency Analysis* [Cheng, 2002], *Optimal Reinsertion* [Moore and Wong, 2003], *Greedy Equivalence Search* [Chickering, 2002], *PC* [Spirtes et al., 2001], and *Max-Min Hill Climbing (MMHC)* [Tsamardinos et al., 2006] algorithms, shows that the $MMHC$ performs best in terms of both correctness of the inferred BNs and and ability to infer large networks [Tsamardinos et al., 2006]. Tsamardinos *et al.* report that building a BN for $5,000$ variables took approximately 14 days, of which 19 hours were spent to infer the skeleton network and the rest 13 days to orient it. This indicates that the state-of-the-art heuristic-based methods for BN structure learning do not scale to large genome domain sizes, leaving this as an open problem.

In this thesis we address the open problems outlined above by devising and implementing run-time and memory efficient parallel algorithms. We also investigate the biological relevance of the learned networks from large mixtures of microarray experiments.

1.2 Concepts of Bayesian Networks and Structure Learning

1.2.1 Markov Assumption and Bayesian Networks

Throughout this thesis, we denote random variables with upper-case letters (e.g. X, X_i) and their values with lower-case letters (e.g. x). We denote sets of random variables with upper-case bold-face letters (e.g. $\mathbf{Z}$) and the corresponding assignment of values for each variable in a given set with a lower-case bold-face letter (e.g. $\mathbf{z}$). Conditional independence of two variables X and Y given set $\mathbf{Z}$ is specified as $\perp\!\!\!\perp$ (e.g. $X \perp\!\!\!\perp Y|\mathbf{Z}$), and conditional dependence as $\not\perp\!\!\!\perp$ (e.g. $X \not\perp\!\!\!\perp Y|\mathbf{Z}$). Let $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ be a set of random variables. In a directed acyclic graph (DAG) on $\mathcal{X}$, a node X_j is called a *descendant* of X_i if a directed path

exists from X_i to X_j, and a *nondescendent* otherwise. Further, a node X_k is called a *parent* of X_i, and X_i is called a *child* of X_k, if there is an edge from X_k to X_i. The set of all parents of X_i is denoted by $Pa(X_i)$. The following is known as the Markov assumption:

Definition 1.1 *Let P be a joint probability distribution over the variables in set $\mathcal{X}$ and $N = (\mathcal{X}, E)$ be a DAG. Then N and P are said to satisfy the Markov assumption if each variable $X_i \in \mathcal{X}$ is conditionally independent of the set of its nondescendants, given the set of its parents [Neapolitan, 2003].*

The pair (N, P) is called a BN if N and P satisfy the Markov assumption [Pearl, 1988]. This allows for a compact representation of the joint probability over $\mathcal{X}$, as the product of the conditional probabilities of each variable given its parents:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid Pa(X_i)).$$

Formally:

Theorem 1.1 *If (N, P) satisfies the Markov assumption, then P is equal to the product of its conditional distributions of all nodes given values of their parents, whenever these conditional distributions exist [Neapolitan, 2003].*

1.2.2 Optimal Structure Learning

When learning the structure of a BN from data, the Bayesian approach utilizes a statistically motivated scoring function to evaluate how well a given graph depicts the dependencies in the given data [Cooper and Herskovits, 1992, Heckerman et al., 1995a, Schwarz, 1978, Lam and Bacchus, 1994, Akaike, 1974]. One example of such a scoring function is the Bayesian

score, where the posterior probability of the proposed graph N is evaluated given the observed data. Suppose we are given m observations of the system, each observation consisting of the value of each of the n random variables in $\mathcal{X}$. These observations can be viewed as an $n \times m$ matrix $D_{n\times m}$, with rows representing variables and columns representing observations. Then, the Bayesian score for a network N given $D_{n\times m}$ is:

$$Score\,(N) = \log P\,(N \mid D)$$

$$= \log P\,(D \mid N) + \log P\,(N) + C,$$

where C is a constant. In this score the two non-constant terms $\log P\,(D \mid N)$ and $\log P\,(N)$ refer respectively to the log-likelihood of the data D given DAG N and the prior probability of N, usually taken to be uniform.

Optimal structure learning is formalized as an optimization problem. Let $Score(N : D_{n\times m})$ be a scoring function which evaluates how well a given DAG N predicts the dependencies in $D_{n\times m}$. Finding an optimal BN structure N_{opt} is equivalent to finding a DAG which optimizes the scoring function:

$$N_{opt} = \underset{N}{\operatorname{argmax}}\; Score\,(N : D_{n\times m})\,.$$

1.2.3 Scoring Functions

To find the optimal network efficiently, it is crucial to choose a scoring function which decomposes into individual score contributions $s\,(X_i, Pa\,(X_i))$ of each of the variables in $\mathcal{X}$ given its parents:

$$Score\,(N) = \sum_{X_i \in \mathcal{X}} s\,(X_i, Pa\,(X_i))\,.$$

Examples of such information theoretic and Bayesian scoring functions include the Bayesian Information Criterion (BIC) [Schwarz, 1978] and Akaike Information Criterion (AIC) [Akaike, 1974], the information theoretic scoring function based on the Minimum Description Length principle (MDL) [Lam and Bacchus, 1994], and Bayesian scores, which estimate the posterior probability of the considered model given the data, such as Bayesian Dirichlet (BD) [Cooper and Herskovits, 1992]) and its extension Bayesian Dirichlet equivalence (BDe) [Heckerman et al., 1995a]. In this work we utilize two of them, namely the MDL and BDe scoring functions, which are described in more detailed below.

Minimum Description length principle-based scoring function

This function is based on the *minimum description length principle*, which states that for given data one should choose such a model to encode the data, which minimizes the description length of both the model itself and the data [Cover and Thomas, 2006]. In the case of Bayesian networks the model is a network (i.e. DAG with corresponding conditional probabilities table). Its description length is decomposed into the sum of description lengths of the component nodes. Because the network describes probability distribution with respect to the input data, it can be used to find optimal encoding of this data, which induces its description length.

Specifically, to describe the graph N, the parents sets of each variable need to be described. This can be achieved by storing the number of parents $|Pa(X_i)|$ and an index of the parent set $Pa(X_i)$ in an agreed upon enumeration of all $\binom{n}{|Pa(X_i)|}$ sets of cardinality $|Pa(X_i)|$. Thus, the description length of the graph N is

$$\sum_{i=1}^{n} \left(\log n + \log \binom{n}{|Pa(X_i)|} \right).$$

Further, to describe the conditional probability distributions the set of all conditional proba-

bility tables (CPT) needs to be saved. Suppose that for variable X_i, r_i denotes the number of discrete states and q_i denotes the number of all possible parent configurations. Then $q_i \cdot (r_i - 1)$ parameters describe the CPT for variable X_i, and given that $\frac{1}{2} \cdot m$ bits are used to represent each parameter, the set of all CPTs can be encoded in:

$$\sum_{i=1}^{n} \frac{1}{2} \cdot q_i (r_i - 1) \log m$$

bits. Finally, the data can be described using conditional entropy $H\left(X_i | Pa\left(X_i\right)\right)$, the description length of which is given by:

$$\sum_{i=1}^{n} m \cdot H\left(X_i | Pa\left(X_i\right)\right).$$

1.2.3.1 Bayesian Dirichlet Equivalence scoring function

The Bayesian Dirichlet Equivalence (BDe) scoring function evaluates the posterior probability of a graph N given the observed data $D_{n \times m}$. The best network is the one which maximizes the posterior probability, as opposed to the MDL score, where the optimal network has minimum description length.

Let r_i denote the number of discrete states of variable X_i, and q_i be the number of possible configurations of its parent set $Pa(X_i)$ in N. Let w_j for $j = 1, \ldots q_i$ be one of these configuration. Let also N_{ijk} be the number of instances in the data set $D_{n \times m}$ where the variable X_i is in state k with parents in configuration j. Thus, N_{ij} is the number of instances where the parents of variable X_i are in configuration j. The BDe score is given by:

$$Score_{BDe}(N : D_N \times m) = \log(p(N)) + \sum_{i=1}^{n}\sum_{j=1}^{q_i} \log\left(\frac{\Gamma(\eta_{ij})}{\Gamma(N_{ij}+\eta_{ij})}\right) + \sum_{k=1}^{r_i} \log\left(\frac{\Gamma(N_{ijk}+\eta_{ijk})}{\Gamma(\eta_{ijk})}\right),$$

where η_{ijk} are the hyperparameter for the Dirichlet prior distribution of the parameters given the network structure N, and $n_{ij} = \sum_{k=1}^{r_i} \eta_{ijk}$. The Gamma function $\Gamma(c) = \int_0^\infty e^{-u}.u^{c-1}\,\mathrm{d}u$ is given by $\Gamma(c) = (c-1)!$ when c is an integer. The hyperparameters are computed as follows:

$$\eta_{ijk} = \eta \times p(x_{ik}, w_{ik}|G_0),$$

where $p(.|G_0)$ represents a probability distribution associated with a *prior Bayesian network* G_0 and η is a parameter representing the equivalent sample size. In our implementation of the BDe score we use the variation BDeu, where a uniform probability is assigned to each configuration of variable state and parent set configuration: $p(x_{ik}, w_{ik}|G_0) = \frac{1}{r_i q_i}$. We also assign uniform distribution to all possible structures, and therefore ignore the $\log(p(N))$ term. The BDe assigns equal scores to graphs from the same Markov equivalence class (see Section 1.2.8).

1.2.4 Faithfulness

Variables X and Y are said to be conditionally independent given set $\mathbf{Z}$ with respect to a probability distribution P if $P(X, Y|\mathbf{Z}) = P(X|\mathbf{Z})P(Y|\mathbf{Z})$. The following condition is an essential assumption we will be making throughout this thesis:

Definition 1.2 *A Bayesian network (N, P) is said to satisfy the faithfulness condition if P embodies all and only independencies that are reflected in the DAG N [Spirtes et al., 2001].*

1.2.5 D-separation

While N encodes some independencies of P directly (e.g. the absence of an edges indicates lack of direct dependence), others are entailed. A graphical criterion which allows for entailed independencies to be read from the BN structure N is called *d-separation*.

Definition 1.3 *Let $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$ be three mutually exclusive subsets of $\mathcal{X}$. Then d-sep($\boldsymbol{X}$, $\boldsymbol{Y}|\boldsymbol{Z}$) denotes that $\boldsymbol{X}$ and $\boldsymbol{Y}$ in N are d-separated by $\boldsymbol{Z}$, and d-sep($\boldsymbol{X}$, $\boldsymbol{Y}|\boldsymbol{Z}$) is said to hold [Neapolitan, 2003] when on every undirected path between a node in $\boldsymbol{X}$ and a node in $\boldsymbol{Y}$, a node Z exists such that either:*

1. *Z does not have two incoming edges and $Z \in \boldsymbol{Z}$, or*

2. *Z does have two incoming edges but neither Z nor any of its descendants in N are in $\boldsymbol{Z}$.*

In a faithful Bayesian network (N, P), *d-sep*$(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$ is equivalent to $\mathbf{X} \perp\!\!\!\perp \mathbf{Y}|\mathbf{Z}$ [Pearl, 1988]. Due to the faithfulness assumption we made in 1.2.4, *d*-separation and independence are used interchangeably.

1.2.6 Constraints-based Learning

In BN structure learning statistical tests or information theoretic criteria are often used to discover conditional independencies. The main idea behind constraint-based BN structure learning consists of determining direct and entailed conditional independencies via statistical tests and *d*-separation relations. The following theorem [Spirtes et al., 2001] is instrumental to the edge discovery procedures in the algorithms discussed in Chapter 4:

Theorem 1.2 *If a probability distribution P is faithful to a DAG N then:*

1. *For each pair of nodes X and Y in $\mathcal{X}$, X and Y are adjacent in N if and only if $X \not\perp\!\!\!\perp Y|\boldsymbol{Z}$ for all $\boldsymbol{Z} \subseteq \mathcal{X}\backslash\{X,Y\}$, and*

2. *For each triplet of nodes X, Y, and Z in N such that X and Z are adjacent, Y and Z are adjacent, but X and Y are not adjacent, $X \rightarrow Z \leftarrow Y$ is a subgraph of N if and only if $X \not\perp\!\!\!\perp Y|\boldsymbol{Z}$ for all $\boldsymbol{Z}$ such that $X, Y \notin \boldsymbol{Z}$ and $Z \in \boldsymbol{Z}$.*

Let $PC(T)$ denote the set of parents and children in N of some variable $T \in \mathcal{X}$. The set is unique for all BNs that are faithful to the same probability distribution P [Verma and Pearl, 1988, 1991].

1.2.7 Markov Boundary

For any variable $T \in \mathcal{X}$, its Markov boundary, denoted $MB(T)$, is defined as any minimal subset of $\mathcal{X}$ which shields T from all other variables in $\mathcal{X}$ excluding T and $MB(T)$:

Definition 1.4 *Markov boundary $MB(T)$ of a variable $T \in \mathcal{X}$ is the minimal set of variables in $\mathcal{X}$ such that $T \perp\!\!\!\perp (\mathcal{X}\backslash (MB(T) \cup \{T\}))\,|MB(T)$.*

For any variable $T \in \mathcal{X}$ the set $MB(T)$ is the union of the parents of T, children of T, and all other parents of children of T (also called *spouses* of T). The set $MB(T)$ is unique for all BNs that are faithful to the same probability distribution P [Pearl, 1988].

1.2.8 Markov Equivalence

Multiple DAGs can represent the same set of conditional independecies, i.e. they have the same d-separations. Such graphs are called Markov equivalent and are formally defined as follows:

Definition 1.5 *Let* $N_1 = (\mathcal{X}, E_1)$ *and* $N_2 = (\mathcal{X}, E_2)$ *be two DAGs containing the same set of nodes* $\mathcal{X}$*. Then* N_1 *and* N_2 *are called* **Markov equivalent** *is for every three mutually disjoint subsets* $\mathbf{A}, \mathbf{B}, \mathbf{C} \subseteq \mathcal{X}$*,* $\mathbf{A}$ *and* $\mathbf{B}$ *are d-separated by* $\mathbf{C}$ *in* N_1 *if and only if* $\mathbf{A}$ *and* $\mathbf{B}$ *are d-separated by* $\mathbf{C}$ *in* N_2 *[Neapolitan, 2003].*

The application of the above definition over graph properties is in probability due to the following theorem:

Theorem 1.3 *Two DAGs are Markov equivalent if and only if, based on the Markov assumption, they entail the same conditional independencies [Neapolitan, 2003].*

Since Markov equivalent graphs represent the same set of independencies, when using Bayesian scoring functions, we will desire that the same score is assigned to all DAGs within the same Markov equivalence class. One such Bayesian score is the Bayesian Dirichlet equivalence (BDe) score.

CHAPTER 2. Parallel Globally Optimal Structure Learning of Bayesian Networks

2.0.9 Previous Work

A major difficulty in BN structure learning is the super exponential search space in the number of random variables. As reported by Robinson [Robinson, 1973], for a set of n variables there exist $\frac{n!2^{\frac{n}{2}(n-1)}}{r \cdot z^n}$ possible DAGs, where $r \approx 0.57436$ and $z \approx 1.4881$. To reduce the search space, marginalization over node orders has been proposed [Koivisto et al., 2004, Ott et al., Silander and Myllymaki, 2006]. Here we briefly review the main idea.

Consider the graph $N = (\mathcal{X}, E)$ of a BN (N, P). Due to its DAG structure, the random variables in $\mathcal{X}$ can be ordered in such a way that for each variable $X_i \in \mathcal{X}$ its parents $Pa\,(X_i)$ precede it in the specified ordering. Optimizing a scoring function for each variable $X_i \in \mathcal{X}$ and all of its respective subsets of preceding elements allows us to discover the optimal network that respects the particular ordering. Further, to investigate all possible graphs we need to consider a total of $n!$ orderings of $\mathcal{X}$. This yields the search space of ordered subsets of $\mathcal{X}$ of size $n!2^n$ (see Figure 2.1). However, as long as $Pa\,(X_i)$ precedes X_i, the order among the nodes in $Pa(X_i)$ is irrelevant. Thus, the search space can be reduced to the 2^n unordered subsets of $\mathcal{X}$.

Even reduced, the search space remains exponential and in practice, exact structure learning without additional constraints has been achieved only for moderate size domains and at

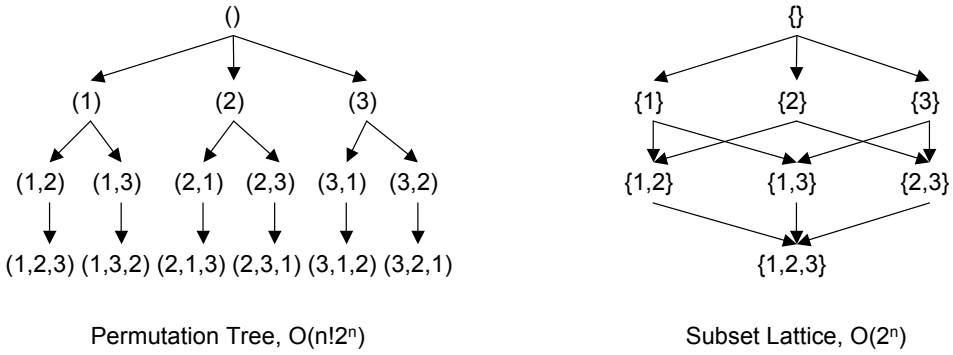


Figure 2.1 A permutation tree and its corresponding subset lattice.

high execution cost. While exact network structure learning, or globally optimal structure learning, has been shown to be NP-hard even with bounded node in-degree [Chickering et al., 1994], constructing exact Bayesian networks without additional assumptions remains valuable nevertheless.

2.0.10 Contributions

In this chapter, we present a parallel algorithm for exact Bayesian network structure learning with optimal parallel efficiency on up to $O\left(\frac{1}{n} \cdot 2^n\right)$ processors regardless of the complexity of the scoring function used. To assess the performance of our algorithms experimentally, we report results on two parallel systems – IBM Blue Gene/P and AMD Opteron cluster with InfiniBand interconnect. Experimental results demonstrate the validity of the theoretical predictions. To our knowledge, the presented work is the first parallel algorithm for exact learning.

The remainder of this chapter is organized as follows: In Section 2.1, we provide a brief overview of the Ott *et al.* sequential method for exact Bayesian network structure learning, upon which our parallel algorithm is based. Sections 2.2 and 2.3 contain our parallel algorithm, and proofs of correctness and run-time and space complexity analysis, respectively. In Sec-

tion 2.4 we present scaling experiments on an IBM Blue Gene/P system and an AMD Opteron InfiniBand cluster by applying our algorithm to the problem of identifying gene regulatory networks, a problem that arises in systems biology.

2.1 Exact Structure Learning of Bayesian Networks

Given a set of observations $D_{n\times m}$ and a decomposable scoring function, the Bayesian network structure learning problem is to find a DAG N on the n random variables that optimizes $Score\,(N)$. Learning the DAG is equivalent to finding the parents of each node in the domain under the acyclicity constraint.

While an exhaustive enumeration of all possible structures is prohibitive in space and time, the notion of ordering the variables (nodes) has been shown to be advantageous [Friedman and Koller, 2003]. Adopting a canonical representation of the DAGs in conjunction with a decomposable scoring function permits a dynamic programming approach, greatly reducing the search space. Exact structure learning algorithms exploit this [Ott et al., Silander and Myllymaki, 2006]. Our parallel algorithm is based on the sequential method of Ott *et al.* [Ott et al.], which we briefly describe here.

Let $X_i \in \mathcal{X}$ and $A \subseteq \mathcal{X} - \{X_i\}$. Using $D_{n\times m}$, a scoring function $s(X_i, A)$ determines the score of choosing A to be the set of parents for X_i; or equivalently, $s(X_i, A)$ specifies how well A predicts X_i given $D_{n\times m}$. For the moment, we do not go into details of how s is computed, except to note that the cumulative s function computations could be the dominant factor in run-time complexity.

Definition 2.1 *Let $X_i \in \mathcal{X}$ and $A \subseteq \mathcal{X} - \{X_i\}$. Let $F(X_i, A)$ denote the highest possible score*

that can be obtained by choosing parents of X_i from A; i.e.,

$$F\left(X_i, A\right) = \max_{B \subseteq A} s(X_i, B).$$

A set B which maximizes the above equation is an optimal parents set for X_i from the candidate parents set A. While $F(X_i, A)$ can be computed by directly evaluating all $2^{|A|}$ subsets of A, it is advantageous to compute it based on the following recursive formulation. For any non-empty set A,

$$F(X_i, A) = \max \begin{cases} s\left(X_i, A\right) \\ \\ \max\limits_{X_j \in A} F\left(X_i, A - \{X_j\}\right). \end{cases}$$

An ordering of a set $A \subseteq \mathcal{X}$ is a permutation $\pi(A)$ of the elements in A. A network on A is said to be consistent with a given ordering $\pi(A)$ if and only if $\forall X_j \in A$, the parents $Pa\left(X_j\right)$ precede X_j in π.

Definition 2.2 *An optimal ordering $\pi^*\left(A\right)$ is an ordering with which an optimal network on A is consistent.*

Let X_j be the last element in $\pi^*\left(A\right)$. Then, the permutation $\pi\left(A - \{X_j\}\right)$ obtained by leaving out the last element X_j is consistent with an optimal network on $A - \{X_j\}$. We write this as:

$$\pi^*\left(A\right) = \pi^*\left(A - \{X_j\}\right) X_j.$$

Definition 2.3 *Given $A \subseteq \mathcal{X}$ and an ordering $\pi(A)$, let $Q\left(A, \pi(A)\right)$ denote the optimal score*

of a network on A, which is consistent with $\pi(A)$*:*

$$Q\left(A, \pi(A)\right) = \sum_{X_j \in A} F\left(X_j, \{X_q \mid X_q \textit{ precedes } X_j \textit{ in } \pi\left(A\right)\}\right).$$

Optimal score of a network on $A \subseteq \mathcal{X}$ is then given by $Q^*(A) = Q\left(A, \pi^*\left(A\right)\right)$. Our goal is to find $\pi^*(\mathcal{X})$ and the optimal network score $Q^*(\mathcal{X})$, and learn the corresponding DAG.

Like the F function, functions π^* and Q can be computed using a recursive formulation. Consider a subset A. To find $\pi^*(A)$, we consider all possible choices for its last element:

$$X_j^* = \underset{X_j \in A}{\operatorname{argmax}}\, Q\left(A, \pi^*(A - \{X_j\})X_j\right)$$

$$= \underset{X_j \in A}{\operatorname{argmax}}\, \left(F\left(X_j, A - \{X_j\}\right) + Q^*\left(A - \{X_j\}\right)\right).$$

Then,

$$Q^*\left(A\right) = F\left(X_j^*, A - \left\{X_j^*\right\}\right) + Q^*\left(A - \left\{X_j^*\right\}\right),$$

$$\pi^*\left(A\right) = \pi^*\left(A - X_j^*\right) X_j^*.$$

By keeping track of the optimal parents sets for each of the variables in $\mathcal{X}$, an optimal network is easily reconstructed.

Using these recursive formulations, a dynamic programming algorithm to compute a globally optimal BN structure can be derived. The algorithm considers all subsets of $\mathcal{X}$ in increasing size order, starting from the empty subset. When considering A, the goal is to compute $F(X_i, A)$ for each $X_i \notin A$ and $Q\left(A, \pi^*(A - \{X_j\})X_j\right)$ for each $X_j \in A$. All the F and Q^* values required for computing these have already been computed when considering subsets $A - \{X_j\}$ for each $X_j \in A$.

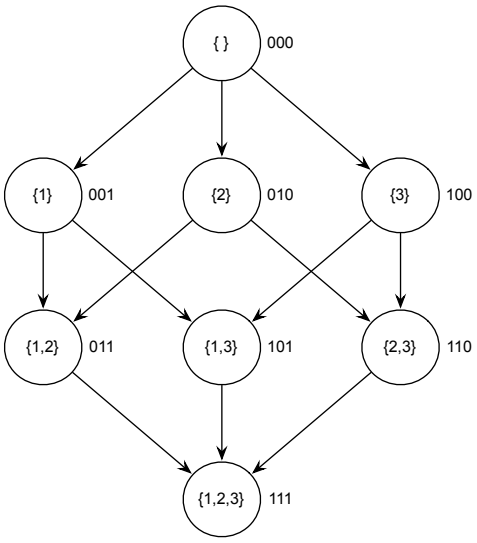


Figure 2.2 A lattice for a domain of size 3 and its partitioning into 4 levels, where the empty set is at level 0. The binary string labels on the right-hand side of each node show the correspondence with a 3-dimensional hypercube.

For a subset A, the F function computations (not including the s function computations required within) take $O((n - |A|) \cdot |A|)$ time and the Q computations take $O(|A|)$ time. The time taken by s function computations depends upon the particular scoring function used. Ignoring them for the moment, the rest of the computations take

$O\left(\Sigma_{i=0}^{n}(n - i + 1) \cdot i \cdot \binom{n}{i}\right) = O(n^2 \cdot 2^n)$ time.

2.2 Parallel Algorithm for Exact Structure Learning

To develop the parallel algorithm, it is helpful to visualize the dynamic programming algorithm as operating on the lattice L formed by the partial order "set inclusion" on the power set of $\mathcal{X}$. Koivisto *et al.* [Koivisto et al., 2004] proved that the permutation tree over all possible orders collapses to the lattice. The lattice L is a directed graph (V, E), where $V = 2^{\mathcal{X}}$ and $(B, A) \in E$ if $B \subset A$ and $|A| = |B| + 1$. The lattice is naturally partitioned into levels, where level $l \in [0, n]$ contains all subsets of size l (see Figure 2.2). A parallel algorithm can

be derived by mapping the nodes to processors and having edges represent communication if the incident nodes are assigned to different processors. A node A at level l has l incoming edges from nodes $A - \{X_j\}$ for each $X_j \in A$, and $n - l$ outgoing edges to nodes $A \cup \{X_i\}$ for each $X_i \notin A$. Functions $Q^*(A)$, $\pi^*(A)$, and a total of $(n-l)$ F functions are computed at node A. All of these values need to be sent along each of the outgoing edges. On an outgoing edge to node $A \cup \{X_i\}$, the $F(X_i, A)$ value is used in computing $Q^*(A \cup \{X_i\})$, and the remaining $F(X_k, A)$ ($X_k \notin A$ and $X_k \neq X_i$) values are used in computing $F(X_k, A \cup \{X_i\})$ values at node $A \cup \{X_i\}$. Note that each of the $(n - l)$ F values at A are used in computing the Q^* value at one of the $n - l$ nodes connected to A by outgoing edges. Each level in the lattice can be computed concurrently, with data flowing from one level to the next.

2.2.1 Mapping to an n-dimensional Hypercube

Observe that the undirected version of L is equivalent to an n-dimensional (n-D) hypercube. Let $(X_1, X_2, \ldots, X_n)$ be an arbitrary ordering of the variables in the domain. A subset A can be represented by an n-bit string ω, where $\omega[j] = 1$ if $X_j \in A$, and $\omega[j] = 0$ otherwise. As lattice edges connect pairs of nodes that differ in the presence of one element, they naturally correspond to hypercube edges (see Figure 2.2). This suggests an obvious parallelization on an n-D hypercube. While we expect the number of processors $p \ll 2^n$, we describe this parallelization in slightly greater detail due to its use as a module in the development of our parallel algorithm.

The n-D hypercube algorithm runs in $n + 1$ steps. Let ω denote the id of a processor and let $\mu(\omega)$ denote the number of 1's in ω. Each processor is active in only one time step – processor ω is active in time step $\mu(\omega)$. It receives $(n - \mu(\omega) + 1)$ F values and one Q^* value and its respective π^* from each of its $\mu(\omega)$ neighbors obtained by inverting one of its 1 bits

to zero. It then computes its own F and Q^* values, and π^*, and sends them to each of its $n - \mu(\omega)$ neighbors obtained by inverting one of its zero bits to 1.

The run-time of step i is $O((n-i+1)\cdot i)$, in addition to the cost of computing the s function on a set of size i, which depends on the application and type of scoring function used. Ignoring the latter, the parallel run-time is $O\left(\sum_{i=0}^{n}(n-i+1)\cdot i\right) = O(n^3)$, using 2^n processors. As the sequential run-time is $O(n^2 \cdot 2^n)$, the parallel efficiency is $\Theta(1/n)$.

2.2.2 Partitioning into k-dimensional Hypercubes

Let $p = 2^k$ be the number of processors, where $k < n$. We assume that the processors can communicate as in a hypercube. This is true of parallel computers connected as a hypercube, hypercubic networks such as the butterfly, multistage interconnection networks such as the Omega, and point-to-point communication models such as the permutation network model or the MPI programming model. Our strategy is to decompose the n-D hypercube into 2^{n-k} k-D hypercubes and map each k-D hypercube to the $p = 2^k$ processors as described previously. Although the efficiency of such a mapping by itself is suboptimal ($\Theta(1/k)$), note that each processor is active in only one time step and $p \ll 2^n$. Therefore, we pipeline the execution of the k-D hypercubes to complete the parallel execution in $2^{n-k} + k$ time steps such that all processors are active except for the first k and last k time steps during the build up and draining of the pipeline.

For convenience of presentation, we use the lattice L and the n-D hypercube interchangeably and use A to denote the lattice node for subset A, and use ω_A to denote the n-bit binary string denoting the corresponding hypercube node. We number the positions of the n-bit binary string using integers 1 through n, with 1 denoting the leftmost position, and n denoting the rightmost position. Let $\omega_A[i, j]$ denote the substring of ω_A between and including positions

i and j. We partition the n-D hypercube into 2^{n-k} k-D hypercubes based on the first $n-k$ bits of node ids. For a lattice node ω_A, $\omega_A[1, n-k]$ specifies the k-D hypercube it is part of and $\omega_A[n-k+1, n]$ specifies the processor it is assigned to. Conversely, a processor with id r computes for all lattice nodes A such that $r = \omega_A[n-k+1, n]$.

2.2.3 Pipelining Hypercubes

Each k-D hypercube is specified by an $(n-k)$ bit string, which is the common prefix to the 2^k lattice/k-D hypercube nodes that are part of this k-D sub-hypercube. The k-D hypercubes are processed in the increasing order of the number of 1's in their bit string specifications, and in lexicographic order within the group of hypercubes with the same number of 1's. Formally, we have the following:

Definition 2.4 *Let H_i and H_j be two k-D hypercubes and let A and B be two nodes in the lattice that map to H_i and H_j, respectively. Then, computation of H_i is initiated before computation of H_j if and only if:*

1. $\mu(\omega_A[1, n-k]) < \mu(\omega_B[1, n-k])$, *or*
2. $\mu(\omega_A[1, n-k]) = \mu(\omega_B[1, n-k])$ *and* $\omega_A[1, n-k]$ *is lexicographically smaller than* $\omega_B[1, n-k]$.

The above total order is used to initiate the 2^{n-k} k-D hypercube evaluations, starting from time step 0. If T denotes the time step in which H_i is initiated and A is a lattice node mapped to H_i, then processor with id $\omega_A[n-k+1, n]$ computes for the lattice node A in time step $T + \mu(\omega_A[n-k+1, n])$.

2.3 Correctness and Complexity

2.3.1 Proof of Correctness

We now prove that when a lattice node is computed, nodes incident to incoming edges have already been computed in previous steps.

Lemma 2.1 *Let (B, A) denote a directed lattice edge. Then, node B is computed at an earlier time step than when node A is computed, and is either 1) local to A's processor, or 2) resides on a different processor but was mapped to the same k-dimensional hypercube as A.*

Proof 1 *(B, A) is a directed lattice edge, and hence $B \subset A$ and $|A| = |B| + 1$. Therefore, ω_A and ω_B differ in one bit position, in which ω_B has '0' and ω_A has '1'. It follows that $\mu(\omega_A) = \mu(\omega_B) + 1$. Let c be the position of the differing bit.*

Case 1: $1 \leq c \leq n - k$

In this case, $\omega_A[n-k+1, n] = \omega_B[n-k+1, n]$. Thus, lattice nodes B and A are mapped to the same processor. As $\mu(\omega_B[1, n-k]) < \mu(\omega_A[1, n-k])$, B is previously computed on the same processor and is available, without requiring any communication, for computing A.

Case 2: $n - k + 1 \leq c \leq n$

In this case, $\omega_A[1, n-k] = \omega_B[1, n-k]$, meaning both A and B are mapped to the same k-D hypercube. Let T be the time step at which computation on this hypercube is initiated.

Then, A is computed in time step $T + \mu(\omega_A[n-k+1, n]) = T + \mu(\omega_B[n-k+1, n]) + 1$, one time step after B is computed.

This proves that the algorithm correctly computes all of the lattice nodes, and hence derives the optimal Bayesian network.

2.3.2 Run-Time Complexity

The parallel algorithm initiates 2^{n-k} k-D hypercubes, one per each time step starting from 0. Each k-D hypercube takes $k+1$ time steps but successive hypercube evaluations are pipelined. Thus, the total number of parallel time steps is given by $2^{n-k} + k$. Ignoring the application-specific scoring function s, the computation of the F functions, Q^*, and the last element of π^* for a lattice node at level l takes $O(l \cdot (n-l+1))$, which is clearly $O(n^2)$. Even with this relaxed analysis, the parallel run-time is $O\left((2^{n-k}+k) \cdot n^2\right)$. Given the serial run-time is $O(n^2 \cdot 2^n)$, optimal parallel run-time is $O(n^2 \cdot 2^{n-k})$, which is achieved by our parallel algorithm when $2^{n-k} > k$. This can be restated as $n > k + \log k$, or $p = 2^k = O\left(\frac{1}{n} \cdot 2^n\right)$. Thus, the parallel algorithm can utilize processors up to exponential in n without sacrificing work-optimality.

The mapping strategy adopted by the parallel algorithm also reduces the number of communications from n to k for each lattice node. Recall that a lattice node A at level l receives l communications in computing its F and Q^* values, and subsequently communicates these values to $n-l$ neighbors, for a total of n communications. Each of these correspond to inverting one of the bits in ω_A. In case of $p = 2^k$ processors, inverting of one of the first $(n-k)$ bits results in a lattice node assigned to the same processor. Only bit inversions in one of the last k bits requires communication with a neighbor. Thus, communication complexity reduces to $O(kn)$ per lattice node, while the computational complexity remains at $O(n^2)$ per lattice node

for the computation of the Q^* and F values. In addition, it is highly likely that the scoring function computation itself dominates the $O(n^2)$ compute time for the rest. Thus, this disparity in computation and communication complexity in favor of relatively lower communication complexity should aid in achieving good scaling in practice.

We now consider the cost of computing $s(X_i, A)$ for all subsets A, and for each $X_i \notin A$. The cost of computing s is particular to the application domain and the specific scoring function used. However, $s(X_i, A)$ is computed from the m observations of the random variables in A and the random variable X_i; thus, the run-time for computing $s(X_i, A)$ is $\Omega(m|A|)$. In the analysis that follows, we consider the scoring function based on the MDL principle, which was utilized in this work, for which computing $s(X_i, A)$ takes $O(m|A|)$ time. Serially, for a subset A, the F computations take $O((n - |A|) \cdot (|A| + m|A|))$ time and the Q computations still take $O(|A|)$ time. The cost of computing F functions, Q^*, and the last element of π^* for all subsets A is given by

$$O\left(\sum_{i=0}^{n} \binom{n}{i} \cdot ((n-i) \cdot (i + m \cdot i) + i)\right) = O\left(m \cdot n^2 \cdot 2^n\right),$$

where $|A| = i$.

In parallel, as described above, the number of parallel time steps is given by $2^{n-k} + k$, and the computation of Q^*, F, and the last element of π^* for a subset A now takes $(n - |A|) \cdot (|A| + m|A|) + |A|$. Thus, the total cost is given by

$$O\left(\sum_{t=0}^{2^{n-k}+k} ((n - |A|) \cdot (|A| + m|A|) + |A|)\right) = O\left(m \cdot n^2 \cdot 2^{n-k}\right).$$

Note that no communication is involved in the computation of s functions, which further widens the complexity gap between computation and communication, aiding the practical scalability

of the algorithm.

2.3.3 Space Complexity

At level $l \in [0, n]$ of the subset lattice, for each of the $\binom{n}{l}$ subsets of size l, the sequential algorithm stores $(n - l)$ F scores and an optimal ordering π^* of size l. Thus a total of

$$\sum_{l=0}^{n} \binom{n}{l} \cdot n$$

elements are stored. As values computed at one level of the lattice are used only at the next level, storing only two levels at a time is sufficient. The maximum number of elements stored at a level occurs when $l = \lceil \frac{n}{2} \rceil$, which using Stirling's formula approximates to $\sqrt{\frac{2}{\pi}} \cdot \frac{1}{\sqrt{n}} \cdot 2^n \cdot n$.

In the parallel algorithm, suppose that processor with id r is computing for a lattice node corresponding to a subset A. Let $i = \mu\left(\omega_A[1, n-k]\right)$. At this node, $n - i - \mu(r)$ F functions and the corresponding $\pi^*(A)$ of size $|A| = i + \mu(r)$ are computed and stored. Each processor stores computed results for $\binom{n-k}{i}$ subsets of size $i + \mu(r)$, for a total of

$$\sum_{i=0}^{n-k} \binom{n-k}{i} \cdot n$$

elements. As before, the maximum storage needed at the same time is given by the maximum value of the summation of two consecutive terms in the above summation. The maximum value of a term occurs when $i = \lceil \frac{n-k}{2} \rceil$, which using Stirling's formula is approximately $\sqrt{\frac{2}{\pi}} \cdot \frac{1}{\sqrt{n-k}} \cdot 2^{n-k} \cdot n$. Thus, the ratio serial/parallel of the maximum number of stored elements yields

$$\frac{\sqrt{n-k}}{\sqrt{n}} \cdot 2^k \geq \frac{1}{\sqrt{2}} \cdot 2^k, \text{ for } k \leq \frac{n}{2}.$$

Therefore, the storage per processor used by the parallel algorithm is within a factor of $\sqrt{2} \approx 1.41$ of the optimal.

2.4 Experimental Results

We developed an implementation of our parallel algorithm in C++ and MPI. To assess its scalability, we used the resulting code to run a series of tests on an IBM Blue Gene/P system and an AMD Opteron InfiniBand cluster. The Blue Gene/P system is comprised of four 850 MHz PowerPC 450 cores and 2 GB main memory per node (512 MB per PowerPC core). All experiments were run with one MPI process per core. For the AMD cluster, we used the TACC Ranger system on the TeraGrid with each node consisting of four AMD Opteron 2.3 GHz quad-core processors and 32 GB memory per node (2 GB per core). All experiments were run with one MPI process per core, with all 16 cores per node sharing the communication channel.

We used the SynTReN package [Van den Bulcke et al., 2006] that synthetically generates observations for the specified number of genes. This easily allows us to change the number of variables, and the number of observations to test the performance with various data sets. As an optimization criterion, we implemented the Minimum Description Length (MDL) score [Lam and Bacchus, 1994], which can be evaluated in linear time with respect to the number of observations in the input data set (reviewed in Section 1.2.3). In terms of the earlier discussion on run-time complexity, the run-time of $MDL(X_i, A)$ is $O(m|A|)$, meeting the lower bound for any scoring function. As scalability improves with higher run-time complexity of the scoring function, choosing a scoring function with the lowest possible run-time complexity also provides the best way to demonstrate the scalability of the algorithm. The quality of the

Table 2.1 Run-times in seconds on the IBM Blue Gene/P (left) and the AMD Opteron InfiniBand cluster (right) for test data with $n = 24$ genes. The number of observations is denoted by m.

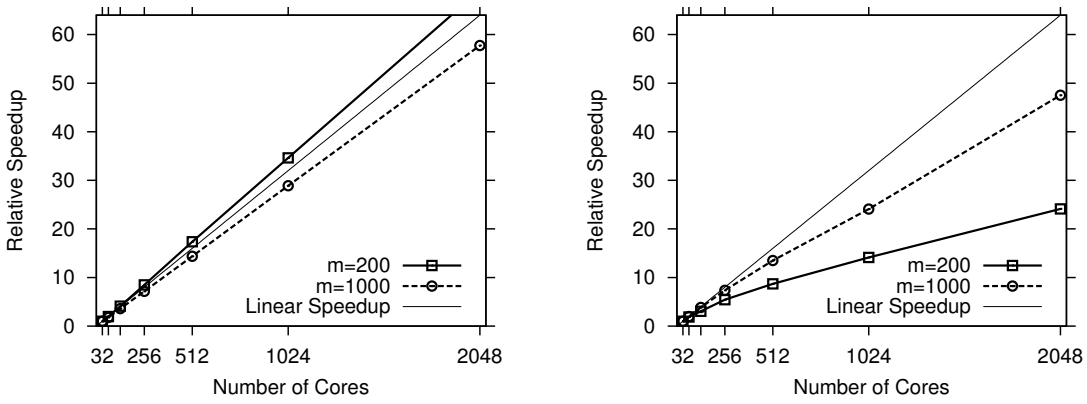
No. Cores	IBM Blue Gene/P		AMD InfiniBand Cluster	
	$m = 200$	$m = 1,000$	$m = 200$	$m = 1,000$
32	1127	4587	209	876
64	578	2590	112	446
128	274	1280	69	229
256	133	641	38.54	119
512	65	319	24.1	65
1024	33	159	14.83	36.39
2048	16	79	8.69	18.43

learned networks is evaluated extensively in Section 2.5.1.

2.4.1 Performance Analysis for Varying Number of Observations

In the first set of experiments we measured how the number of observations influences the scalability of our solution, by fixing the number of genes. A smaller number of observations lowers the computational complexity, but leaves the communication complexity unaffected. This creates the possibility that the cost of the communication dominates, which can limit scalability. However, the observations need to grow as a function of the number of variables, to be able to derive a meaningful network. To run the experiments we used two data sets consisting of $n = 24$ genes with $m = 200$ and $m = 1,000$ observations, respectively. Run-times measured as a function of the number of processors and corresponding relative speedups on the IBM Blue Gene/P and the AMD Opteron InfiniBand cluster are presented in Table 2.1 and Figure 2.3, respectively.

The execution time increases linearly with the number of observations, and at the same time our implementation on the Blue Gene/P maintains close to linear scalability up to $2,048$ cores. The super linear speedup for the data set with 200 observations can be explained



(a) Blue Gene/P

(b) AMD Opteron InfiniBand Cluster

Figure 2.3 Relative speedup as compared to a 32-core run for test data with $n = 24$ on (a) IBM Blue Gene/P and (b) AMD Opteron InfiniBand cluster.

with cache effects. These results indicate that computation is the dominant factor, even for small number of observations. The results on the AMD cluster are not as good due to the differences in the performance of the interconnection network. The Blue Gene/P network has low latency. In addition, 16 MPI processes running on one node of the AMD cluster share the same InfiniBand port, diluting the bandwidth available per core. The results for $m = 1,000$ on the AMD cluster exhibit much better scaling than for $m = 200$. This is due to the linear increase in computation as a function of m, while the communication remains unaffected.

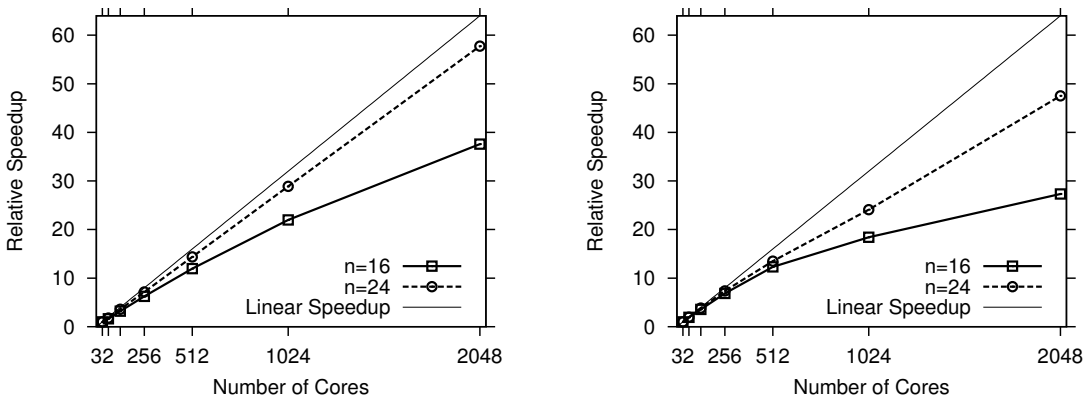
2.4.2 Performance Analysis for Varying Number of Genes

In the second set of experiments we studied the performance of the parallel algorithm with respect to the number of input variables. We processed a collection of data sets with fixed number of observations $m = 1,000$, and varying number of genes n, and recorded the resulting run-times. Obtained results are summarized in Table 2.2 and Figure 2.4, respectively.

The run-times are reflective of the exponential dependence on n. Note that memory required also grows exponentially with n, even though the entire lattice need not be stored

Table 2.2 Run-time results in seconds on the IBM Blue Gene/P (left) and the AMD Opteron InfiniBand cluster (right) for test data with $m = 1,000$ observations. The number of genes is denoted by n.

No. Cores	IBM Blue Gene/P		AMD InfiniBand Cluster	
	$n = 16$	$n = 24$	$n = 16$	$n = 24$
32	13.5	4587	2.24	876
64	8.32	2590	1.16	446
128	4.21	1280	0.63	229
256	2.16	641	0.33	119
512	1.13	319	0.18	65
1024	0.61	159	0.12	36.39
2048	0.36	79	0.08	18.43



(a) Blue Gene/P

(b) AMD Opteron InfiniBand Cluster

Figure 2.4 Relative speedup as compared to a 32-core run for the test data with $m = 1,000$ on (a) IBM Blue Gene/P and (b) AMD Opteron InfiniBand cluster.

throughout. Thus, going parallel is advantageous both from the perspective of run-time and the ability to solve larger problems. Using 2,048 cores, our implementation was able to learn the structure of a BN for 33 genes in 1 hour and 14 minutes. This task would require days if executed on a sequential computer [Ott et al.].

2.5 Biological Validation and Application to *Arabidopsis Thaliana*

Further we proceed to assess the quality of the learned networks using both synthetic and real biological gene expression data.

2.5.1 Synthetic Validation using SynTReN

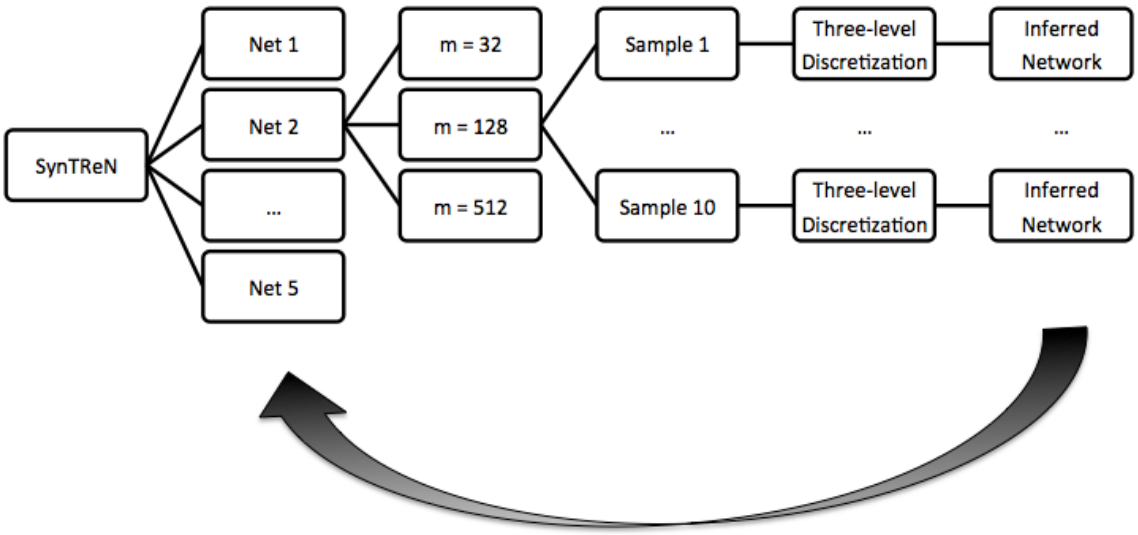


Figure 2.5 Diagram of synthetic data generation procedure using *SynTReN*.

SynTReN is a software package for synthetic gene expression data generation, which allows us to control the number of genes (n) as well as observations (m) in the generated datasets [Van den Bulcke et al., 2006]. We used *SynTReN* to generate 5 gene networks

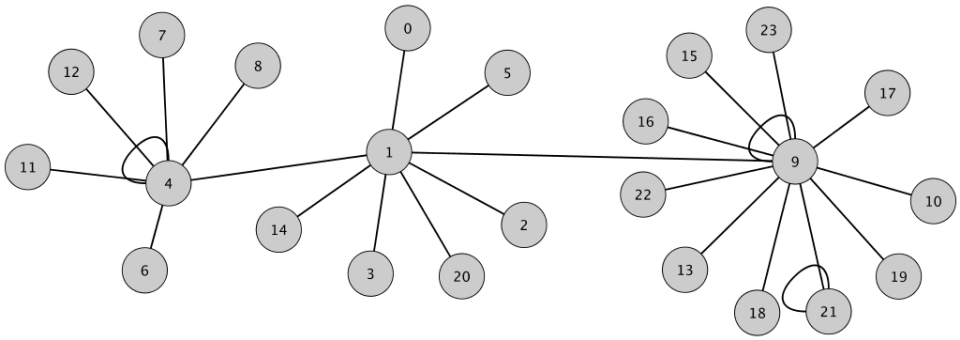


Figure 2.6 Network 1 of the five synthetically generated gene networks of 24 genes. Three self-loops are present at nodes 4, 9, and 21, respectively.

(biological noise: 0.25, experimental noise: 0.01, input noise: 0.01, percent activators: 0.5, correlation noise: 0.1, subnetwork selection: cluster addition). Self-loops were introduced in some of the networks to test how they affect the quality of the learned networks (see Figure 2.6). To assess performance over a range of parameters, we consider three different m values: $m = 32$, 128, and 512 (Figure 2.5). For each of the five networks, 10 datasets of sizes $n = 24$ and respectively $m = 32$, 128, and 512 were generated, corresponding to data matrices: $D_{24\times32}$, $D_{24\times128}$, and $D_{24\times512}$ (total of $5 \times 3 \times 10 = 150$ datasets). Each data-matrix was subjected to standardization and discretization to three levels corresponding to under-expressed (< -0.75), over-expressed (≥ 0.75), and no change (remaining values). For each resulting discretized data matrix a network was induced. Experiments were performed to compare correctness of networks learned using the MDL vs BDe scoring functions, the exact vs a heuristic Bayesian network structure learning method as implemented in the software package *Banjo* [Hartemink], as well as the exact vs an information theoretic approach as implemented by the software tool *TINGe* [Zola et al., 2010]. The quality of the networks is also assessed as a function of the number of observations m.

To evaluate the correctness of the learned networks, we compare them to the "ground

Table 2.3 Exact structure learning with MDL score. Sample size is denoted by m, and for each m, lines 1 through 5 report for networks 1 through 5, respectively; $--$ indicate empty networks.

$m = 32$

TP	FP	TN	FN	SEN	$SPEC$	$PREC$	F-score
–	–	–	–	–	–	–	–
2.00	0.00	251.00	23.00	0.08	1.00	1.00	0.15
1.60	1.00	252.00	21.40	0.07	1.00	0.62	0.12
1.00	0.10	247.90	27.00	0.04	1.00	0.95	0.07
1.13	0.50	252.50	21.88	0.05	1.00	0.77	0.09

$m = 128$

TP	FP	TN	FN	SEN	$SPEC$	$PREC$	F-score
1.00	1.67	251.33	22.00	0.04	0.99	0.39	0.08
3.50	2.30	248.70	21.50	0.14	0.99	0.62	0.23
2.80	1.70	251.30	20.20	0.12	0.99	0.64	0.20
2.00	3.30	244.70	26.00	0.07	0.99	0.38	0.12
3.70	0.80	252.20	19.30	0.16	1.00	0.82	0.27

$m = 512$

TP	FP	TN	FN	SEN	$SPEC$	$PREC$	F-score
10.60	2.90	250.10	12.40	0.46	0.99	0.78	0.58
9.60	10.00	241.00	15.40	0.38	0.96	0.49	0.43
9.10	9.80	243.20	13.90	0.40	0.96	0.48	0.43
10.60	6.50	241.50	17.40	0.38	0.97	0.62	0.47
10.80	5.20	247.80	12.20	0.47	0.98	0.68	0.55

truth" networks generated by $SynTReN$. Directionality can accurately be inferred only in presence of intervention data (e.g. knocking out a gene, over-expressing a gene). Because $SynTReN$ does not allow for the generation of such data, to avoid penalizing networks which belong to the same Markov equivalent class, and therefore are indistinguishable by Bayesian scoring functions (reviewed in Sections 1.2.3 and 1.2.8), we ignore directions and compare the networks as undirected. We report true positives (TP), false positives (FP), true negatives (TN), false negatives (TN), and compute recall (sensitivity) $\left[\frac{TP}{TP+FN}\right]$, specificity $\left[\frac{TN}{TN+FP}\right]$, precision $\left[\frac{TP}{TP+FP}\right]$, and F-score $\left[2 \cdot \frac{precision \cdot recall}{precision+recal}\right]$. The results for each network (average over 10 samples for each m) are summarized in the Tables below.

Table 2.4 Exact structure learning with BDe score. Sample size is denoted by m, and for each m, lines 1 through 5 report for networks 1 through 5, respectively.

$m = 32$							Exact (BDe)	Exact (MDL)
TP	FP	TN	FN	SEN	$SPEC$	$PREC$	F-score	F-score
7.5	37.2	215.8	15.5	0.33	0.85	0.17	0.22	–
9.70	34.1	216.9	15.3	0.39	0.86	0.22	0.28	0.15
6.50	38.5	214.5	16.5	0.28	0.85	0.14	0.19	0.12
8.30	38.9	209.1	19.7	0.3	0.84	0.18	0.22	0.07
9.10	31.6	221.4	13.9	0.4	0.88	0.23	0.29	0.09
$m = 128$								
TP	FP	TN	FN	SEN	$SPEC$	$PREC$	F-score	F-score
12.30	11	242	10.7	0.53	0.96	0.53	0.53	0.08
11.40	21.4	229.6	13.6	0.46	0.91	0.35	0.4	0.23
11.60	25.9	227.1	11.4	0.5	0.9	0.31	0.38	0.2
8.80	23.5	224.5	19.2	0.31	0.91	0.27	0.29	0.12
12.30	17.1	235.9	10.7	0.53	0.93	0.42	0.47	0.27
$m = 512$								
TP	FP	TN	FN	SEN	$SPEC$	$PREC$	F-score	F-score
16.40	10.9	242.1	6.6	0.71	0.96	0.6	0.65	0.58
19.40	20.1	230.9	5.6	0.78	0.92	0.49	0.6	0.43
18.40	23.7	229.3	4.6	0.8	0.91	0.44	0.57	0.43
16.40	25.6	222.4	11.6	0.59	0.9	0.39	0.47	0.47
19.10	16.8	236.2	3.9	0.83	0.93	0.53	0.65	0.55

2.5.1.1 Exact Learning: MDL vs BDe Scores

We compare the quality of networks learned using the MDL and BDe scores, and report the summary statistics in Tables 2.3 and 2.4, respectively. Our results indicate that sample size of $m = 32$ is insufficient for the MDL scoring function to make meaningful predictions. In particular, for network 1 when $m = 32$ the algorithm learned empty networks (all variables were independent). In the case of $m = 128$, while the TP rates are low as compared to the BDe results, the FP rates are also much lower than those of BDe networks. Overall the MDL scoring function is more conservative than the BDe scoring function, in that it prefers graphs with fewer edges, as is expected in theory [Bouckaert, 1995].

2.5.1.2 Heuristic (Banjo) vs Exact with BDe Score

We compare the quality of the networks learned using the exact algorithm to those learned using a heuristic algorithm. To generate the latter, we use a well-established software *Banjo* developed by Hartemink [Hartemink], which implements simulated annealing with BDe scoring function (equivalent sample size: 1, and maximum parents: 7). Without bounding the node in-degree our algorithm learns a network of 24 variables and 200 and 1000 observations in 8.69s and 18.43s, respectively (Table 2.1). We let the heuristic algorithm run for 30s in all cases. Note that limiting the number of parents effectively reduces the search space of the heuristic algorithm (imposed by implementation). Since in the true networks, no variable has more than 3 parents, this limitation is favorable to the performance of the heuristic algorithm as it decreases the possible DAGs in the search space. The summary statistics computed by the exact algorithm and *Banjo* are given in Tables 2.4 and 2.5, respectively.

A comparison of the average F-scores between the two algorithms for all networks and sample sizes is given in Figure 2.7. It can be seen that on average, although the results are close, the exact algorithm generally performs better than the heuristic algorithm, with the exception of the case for $m = 32$ and network 1. We find this result interesting, as the average BDe scores over the 10 samples computed by the exact algorithm and *Banjo* in this case were -620.25 and -624.30, respectively. This indicates that the exact algorithm does find a "more optimal" than the heuristic algorithm network, w.r.t. to the BDe score. However, in this case, this does not depict the biological structure more closely. Recall that network 1 has three self-loops. We believe that these self-loops (Figure 2.6) are a possible cause for this outcome. To allow for modeling of loops different models (e.g. dynamic Bayesian networks), or a combination of models, could be considered.

Table 2.5 Heuristic structure learning with BDe score (Banjo). Sample size is denoted by m, and for each m, lines 1 through 5 report for networks 1 through 5, respectively.

							Banjo	Exact (BDe)
$m = 32$								
TP	FP	TN	FN	SEN	$SPEC$	$PREC$	F-score	F-score
7.56	34	219	15.44	0.33	0.87	0.18	0.23	0.22
8.44	35.44	215.56	16.56	0.34	0.86	0.19	0.24	0.28
7	42.89	210.11	16	0.3	0.83	0.14	0.19	0.19
8.1	40.9	207.1	19.9	0.29	0.84	0.16	0.21	0.22
9.2	30.8	222.2	13.8	0.4	0.88	0.23	0.29	0.29
$m = 128$								
TP	FP	TN	FN	SEN	$SPEC$	$PREC$	F-score	F-score
12	15	238	11	0.52	0.94	0.45	0.48	0.53
11.1	24.7	226.3	13.9	0.44	0.9	0.31	0.37	0.4
10.8	29.3	223.7	12.2	0.47	0.88	0.27	0.34	0.38
8.75	25.75	222.25	19.25	0.31	0.9	0.25	0.28	0.29
12.6	19.8	233.2	10.4	0.55	0.92	0.39	0.45	0.47
$m = 512$								
TP	FP	TN	FN	SEN	$SPEC$	$PREC$	F-score	F-score
16.5	14.7	238.3	6.5	0.72	0.94	0.53	0.61	0.65
19	24.1	226.9	6	0.76	0.9	0.44	0.56	0.6
18.8	27.1	225.9	4.2	0.82	0.89	0.41	0.55	0.57
15.8	30.1	217.9	12.2	0.56	0.88	0.34	0.43	0.47
19.7	20.2	232.8	3.3	0.86	0.92	0.49	0.63	0.65

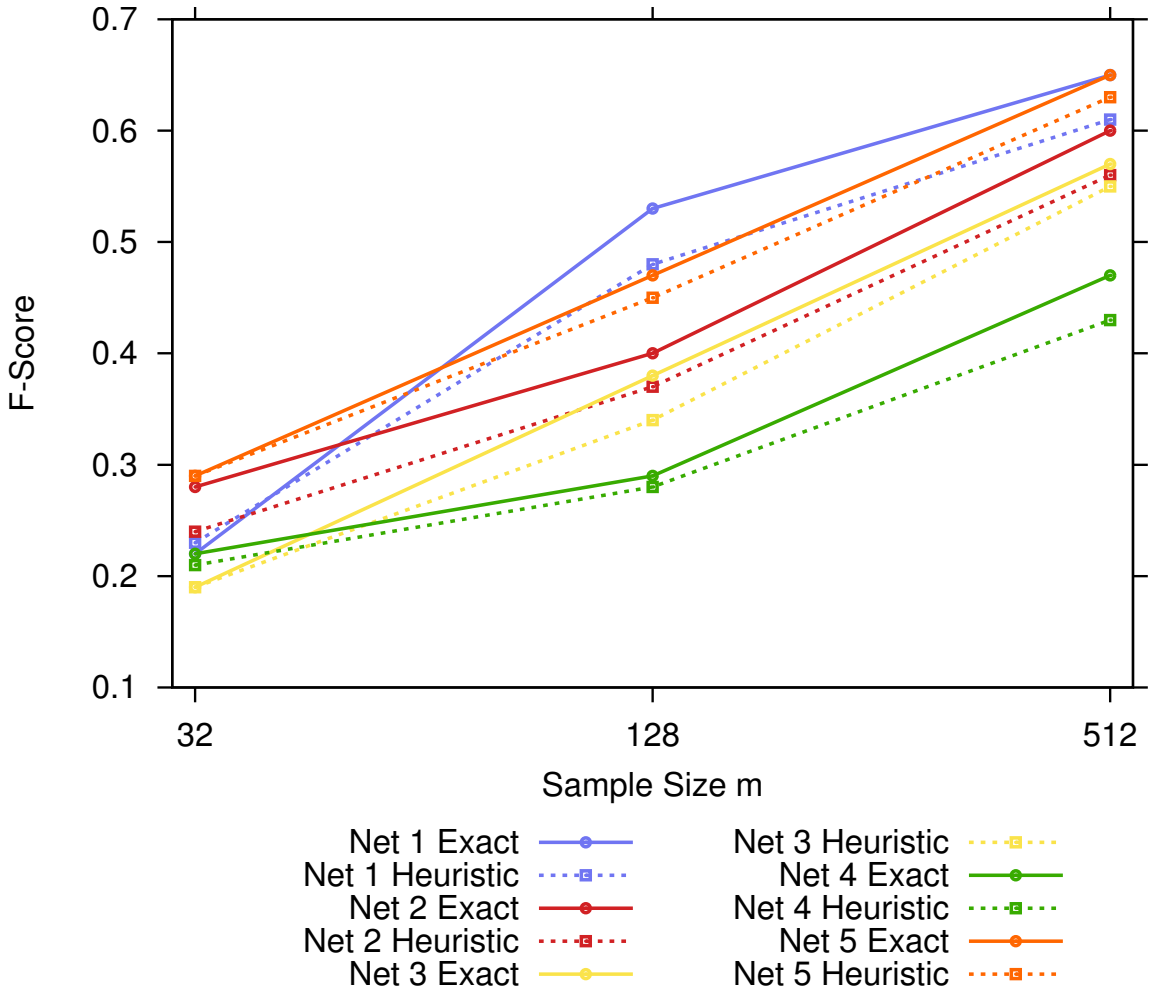


Figure 2.7 Comparison of the average F-scores $\left[2 \cdot \frac{precision \cdot recall}{precision + recal}\right]$ over 10 samples for networks learned with the exact (solid lines) and heuristic (dashed lines) algorithms. The connection between the average score values are gives solely to illustrate the trend. Results for each network are color-coded in blue(network 1), red (network 2), yellow (network 3), green (network 4), and orange (network 5).

Table 2.6 Information theoretic approach (TINGe). Sample size is denoted by m, and for each m, lines 1 through 5 report for networks 1 through 5, respectively.

							TINGe	Exact (BDe)
$m = 32$								
TP	FP	TN	FN	SEN	$SPEC$	$PREC$	F-score	F-score
15.8	43.6	209.4	7.2	0.69	0.83	0.27	0.38	0.22
19.1	74.8	176.2	5.9	0.76	0.7	0.21	0.33	0.28
21.6	118.9	134.1	1.4	0.94	0.53	0.16	0.27	0.19
19	79.4	168.6	9	0.68	0.68	0.2	0.31	0.22
18	43.6	209.4	5	0.78	0.83	0.3	0.43	0.29
$m = 128$								
TP	FP	TN	FN	SEN	$SPEC$	$PREC$	F-score	F-score
20.1	26.9	226.1	2.9	0.87	0.89	0.43	0.58	0.53
21.3	81.6	169.4	3.7	0.85	0.67	0.21	0.33	0.4
23	132.2	120.8	0	1	0.48	0.15	0.26	0.38
19.4	72	176	8.6	0.69	0.71	0.22	0.33	0.29
22.8	61.2	191.8	0.2	0.99	0.76	0.27	0.43	0.47
$m = 512$								
TP	FP	TN	FN	SEN	$SPEC$	$PREC$	F-score	F-score
21.3	22.2	230.8	1.7	0.93	0.91	0.49	0.64	0.65
21.1	82.7	168.3	3.9	0.84	0.67	0.2	0.33	0.6
23	134.2	118.8	0	1	0.47	0.15	0.26	0.57
21.4	63.4	184.6	6.6	0.76	0.74	0.25	0.38	0.47
23	64.5	188.5	0	1	0.75	0.26	0.42	0.65

2.5.1.3 Exact BN Learning with BDe Score vs Information Theoretic Approach (TINGe)

Finally, we compare the quality of the networks learned by the exact BN algorithm with BDe score to an information theoretic approach, implemented in *TINGe* [Zola et al., 2010]. *TINGe* first computes pairwise mutual information between every pair of genes, to create an initial network, and later uses a post-processing procedure, called Data Processing Inequality (DPI), which attempts to remove indirect interactions. The summary statistics computed for *TINGe* are given in Tables 2.6.

We draw attention to the comparison of the F-scores of both approaches. When the

number observations is small ($m = 32$), the information theoretic approach performs better. However, as m increases, the exact BN learning improves and outperforms the information theoretic approach. This is an interesting observation, in that combining the two approaches and applying them based on the number of available observations could yield the best of both.

2.5.1.4 Concluding Remarks on Synthetic Data Validation

In conclusion, we observed that in all cases, including different scoring functions and different approaches, the quality of the networks improves consistently as the number of observations m increases. When compared to exact learning with MDL score, the BDe score exact learning produces more accurate networks. Exact learning with BDe score also performs better than heuristic BN learning, as implemented in *Banjo*. Finally, an information theoretic approach which uses DPI to remove indirect interactions ($TINGe$), performs better than exact BN learning with BDe score when only a few observations are available. However, the exact BN approach with BDe outperforms $TINGe$ for larger values of m. Thus, exact BN structure learning with BDe score produces the most accurate networks, provided that sufficient number of observations are available.

2.5.2 Application to the Carotenoid Biosynthesis Pathway in *Arabidopsis Thaliana*

To evaluate the biological relevance of the networks learned by the exact algorithm for BN structure learning, we applied our software to a set of genes known to be involved in the Carotenoid Biosynthesis pathway in *Arabidospis thaliana*.

Carotenoids are essential in the human diet as a major source of vitamin A (provitamine A carotenoids), antioxidant activity, and limiting age-related degeneration of the eye [Cazzonelli et al., 2010]. In plants, the colors resulting from carotenoid pigments attract insects for pol-

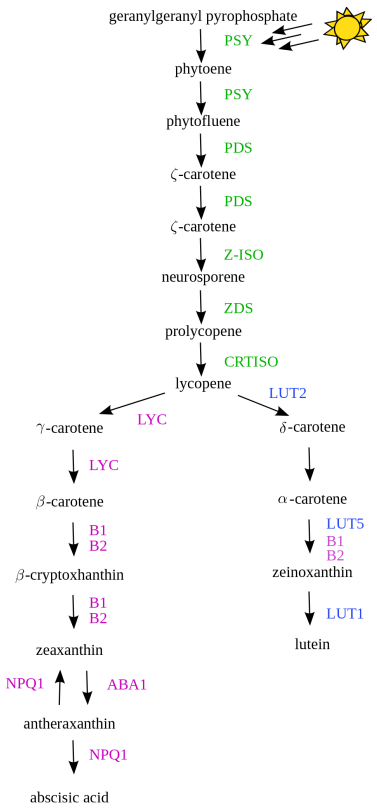


Figure 2.8 Diagram of the Key Steps in the Carotenoid Biosynthesis in *Arabidopsis*.

lination and seed dispersal. Carotenoids are required for photosynthesis, light harvesting and photoprotection, as well as, for the production of shoot and root growth hormone ABA (See Figure 2.8).

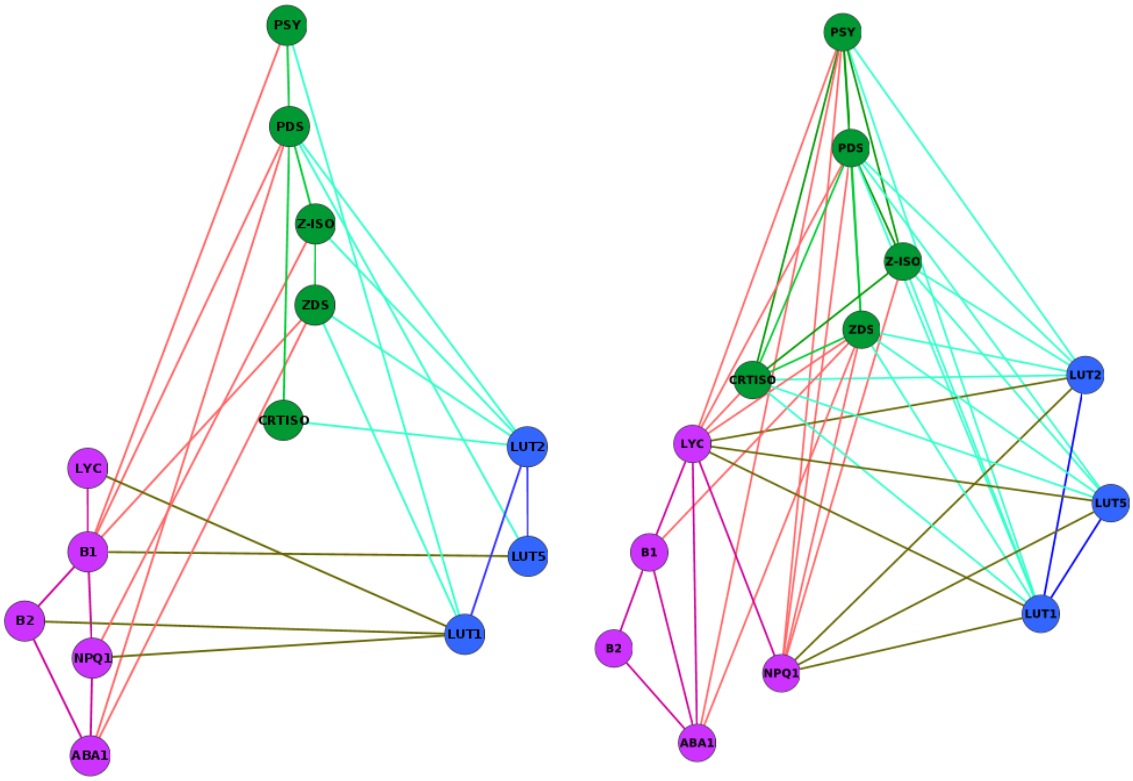
While the carotenoid biosynthesis pathway is actively studied, it is not yet fully understood. Two steps are considered key for the regulation of the pathway. The first one is the light-responsive enzyme PSY catalyzing the synthesis of phytoene from geranylgeranyl pyrophosphate. The second one is considered to be the bifurcation point after lycopene, to produce epsilon- and beta-carotenoids (left- and right-hand branches of the pathway in Figure 2.8, respectively). We will use this information as prior knowledge in our attempt to evaluate the biological relevance of the learned transcription network.

2.5.2.1 Data Preparation

A list of 13 genes (*Arabidopsis* Gene Identifiers: AT1G06820, AT1G10830, AT3G04870, AT1G08550, AT1G31800, AT3G10230, AT3G53130, AT4G14210, AT4G25700, AT5G17230, AT5G52570, AT5G57030, AT5G67030) known to be involved in the carotenoid biosynthesis pathway was downloaded from the Plant Metabolic Network portal [PMN]. Two different sets of experimental conditions were collected from the NASCArray [NAS, 2009], AtGenExpress [TAI, 2009], ArrayExpress [Arr, 2009], and GEO [GEO, 2009] repositories, preprocessed, and analyzed: 1) targeted light experiments, and 2) all available microarray chips (all conditions). For the latter dataset (all experiments) we used the dataset generated and preprocessed for whole genome network analysis by Aluru *et.* al. [Aluru et al., Under Review]. This resulted in a collection of $3,137$ chips. For the former dataset, 12 experiments were hand picked and collected from the NASCArrays [NAS, 2009] and AtGenExpress [TAI, 2009] repositories. After preprocessing, this dataset contained 508 chips. For pre-processing, we followed the steps in quality control, normalization, and data filtering described in Aluru *et.* al. [Aluru et al., Under Review]. We briefly review them here for completeness.

Data Preprocessing

A total of $3,546$ non-redundant raw CEL files were collected initially. Various quality controls metrics were examined, such as BioB spike-in transcript and scaling factor (removed if fell outside 3-fold of mean scale factor or did not call BioB present). Outlier chips were disregarded using the affyPLM Bioconductor library (removed chips which had relative log expression spread around zero greater than 0.75 IQR, or normalized unscaled standard errors spread around one greater than 0.075 IQR). Affymetrix probe intensities were converted to expression values using the MAS 5.0 procedure (scaling factor of 1000), log_2 transformed and



(a) Bayesian Network (b) Pairwise MI-based Network

Figure 2.9 Gene networks generated using the 2.9(a) BN model and 2.9(b) MI-based model under light experimental conditions.

normalized using quantile normalization. Finally, genes with expression profile IQR lower than 0.65 were discarded.

2.5.2.2 Networks Generation and Discussion

We used our software, *ParaBayL*, to learn a BN structure from the two data matrices of 508 light- and 3,137 all-experimental conditions. We also used *TINGe* which computes a co-expression network based on pairwise mutual information and uses Data Processing Inequality (DPI) to remove indirect interactions (parameters: p-value for DPI of 0.001) [Zola et al., 2010].

We color-coded the genes corresponding to each of the three branches of the pathway

with green for the primary branch, purple for the beta- and blue for the epsilon-carotenoids branches. We also color-coded the edges to indicate interactions between genes from the different branches, with red for edges between primary and beta-carotenoids branches, tiel for edges between primary and epsilon-carotenoids branches, and olive for edges between the epsilon- and beta-carotenoid branches. We caution the reader that the generated networks in Figure 2.9 were learned from transcription data alone and that the edges have different meaning from the edges in the metabolic pathway depicted in Figure 2.8. Thus, we do not expect an exact match between the two diagrams; the expectation is that genes with similar function will form close modules in the transcription network.

Targeted Light Experimental Conditions

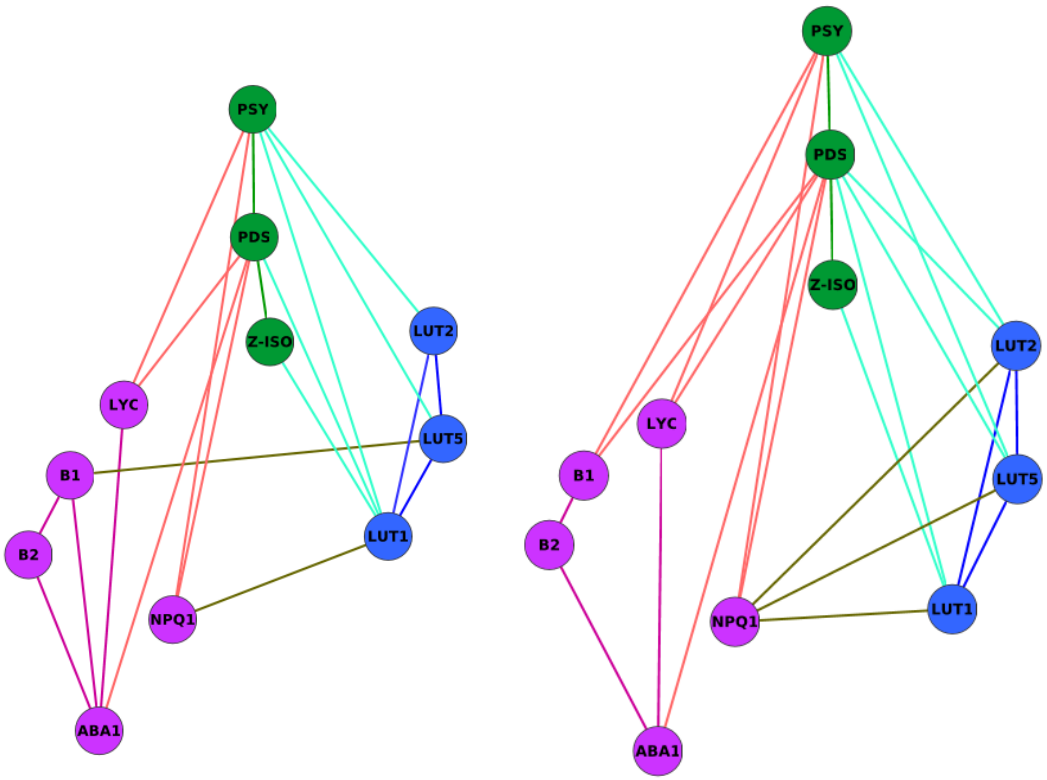
The resulting networks under light conditions are shown in Figure 2.9. Several observations can be made from Figure 2.9(a). First, we notice that the genes corresponding to the epsilon-carotenoids and those corresponding to the beta-carotenoids branch, are interconnected within their respective branches and form neat modules. The genes corresponding to the primary branch of the metabolic pathway are also interconnected. Interestingly, we notice that the edges PSY–PDS, PDS–Z-ISO, Z-ISO–ZDS connect genes corresponding to enzymes required for *consecutive reactions* in the metabolic pathway. Similarly, in the other longer branch of the beta-carotenoids, we have the edges LYC–B1 (B2), B1–NPQ1, B2–ABA1, where both B1 and B2 catalyze the β-carotene to zeaxanthin reactions. Further consideration of the edges between the beta-and epsilon-carotenoids branches (olive color), reveals that an edges between B1 and B2 on one hand and LUT5 on another, are to be expected as B1 and B2 together with LUT5 catalyze the reaction α-carotene $\rightarrow$ zeinoxanthin, and therefore they are all likely to be co-regulated. In the learned BN network, we find an edge between B1 and LUT5. It is

possible that the rest of the olive edges might be an indirect mitigation of this co-regulation, but there is no hard evidence to make this claim. Finally, we notice that on average, there are more edges present between the primary branch and the beta-carotenoid branch (tiel edges), and also the primary branch and the epsilon-carotenoids branch (red edges), than between the remaining two branches (olive edges), which could be due to the regulatory role of the primary branch downstream. Again, this is only a speculation, and no firm claims can be made until further biological validation is carried out.

In comparison, the pairwise MI-based network depicted in Figure 2.9(b) from the same data appears more dense with a total of 48 edges as compared to 28 in the Bayesian network model ($\approx 42\%$ more edges). Overall, in this network as well the beta-carotenoid branch genes are interconnected but the NPQ1 gene is only connected to LYC. Similarly, the genes in the primary pathway are also interconnected, however the linearity we observed in the Bayesian network is not as apparent, with PDS connecting directly to ZDS instead of through Z-ISO. One important connection that does appear in this network is a link between CRTISO from the primary branch and LYC. This is one edge we would have expected to see in the Bayesian network as well; however it appears to be mitigated via other edges indirectly. Although the pairwise MI-based network does call quite a few more edges than the Bayesian network, with the potential of higher FP.

All Experimental Conditions

We note that in the filtering step of the preprocessing pipeline, 2 genes (ZDS, CRTISO) were removed from the initial set of 13 in the all-experiments data set. The resulting networks are shown in Figure 2.10. In the presence of all experimental conditions, many edges are resolved with the Bayesian model resulting in a total of 21 edges and the MI-based model



(a) Bayesian Network (b) Pairwise MI-based Network

Figure 2.10 Gene networks generated using the 2.10(a) BN model and 2.10(b) MI-based model under light experimental conditions.

with 24 (exactly half of the 48 edges under light conditions only). The two methods also appear to agree on most of the predicted edges. In both networks, we see a nice module for the luteins, and the same linearity trend in the primary branch (PSY–PDS–Z-ISO), which does correspond to order in which the metabolic reactions occur, as discussed above. The beta-carotenoid branch genes in both networks now are not as tightly linked, and LYC in both networks links directly to PDS and PSY.

CHAPTER 3. Parallel Globally Optimal Structure Learning of Bayesian Networks with Bounded Node In-degree

We investigate structure learning with user-specified bounded in-degree d, which is the maximum allowed in-degree (or equivalently, the number of parents) of any node in the network. We show that for $d < \frac{1}{3}n - \log mn$ the asymptotic run-time complexity remains unaffected as a function of d. An important consequence of this result is that networks with a generous allowance of bounded in-degree can be learnt in the same time as networks where only constant in-degree is allowed. We also show that $d \geq \lceil \frac{n}{2} \rceil$ results in the same asymptotic run-time complexity as $d = n - 1$, where no restriction is placed on the in-degree.

3.1 Structure Learning with Bounded Node In-degree

As noted before, the run-time complexity of exact structure learning is dominated by the total cost of computing the $s(X_i, A)$ functions, even when the cost is equal to the lower bound of $\Omega(m|A|)$. This raises the possibility of designing a faster algorithm by restricting the maximum in-degree, or equivalently the number of parents, of any node in the network. Given a pre-specified maximum node in-degree d, the BN structure can be learned more efficiently by only considering subsets of size at most d as potential parents sets. We refer to this as the *bounded in-degree* criterion. This is a reasonable assumption as it is unlikely that any node will have all other nodes as parents. In this Section we first describe modifications to the

general algorithm presented in Section 2.2 for exploiting bounded in-degree, and subsequently characterize run-time complexity as a function of the degree d.

3.1.1 Algorithm for Bounded In-degree

Definition 3.1 *Let $X_i \in \mathcal{X}$ and $A \subseteq \mathcal{X} - \{X_i\}$. Let F_d denote the highest possible score that can be obtained by choosing no more than d parents from A, for some pre-specified integer $d < n$.*

$$F_d\left(X_i, A\right) = \max_{B \subseteq A,\ |B| \leq d} s(X_i, B).$$

For a non-empty set A, F_d can be computed based on the following recursive formulation:

$$F_d\left(X_i, A\right) = \begin{cases} F\left(X_i, A\right), & \text{if } |A| \leq d \\ \max\limits_{X_j \in A} F_d\left(X_i, A - \{X_j\}\right), & \text{otherwise} \end{cases}$$

$$= \max \begin{cases} s\left(X_i, A\right), & \text{if } |A| \leq d \\ \max\limits_{X_j \in A} F_d\left(X_i, A - \{X_j\}\right). \end{cases}$$

Using the above definition to compute the optimal parent sets under the maximum in-degree constraint, an optimal ordering $\pi_d^*\left(A\right)$ is an ordering with which an optimal network on A with no more than d parents per node is consistent. The following definition of optimal network score is employed:

Definition 3.2 *Given $A \subseteq \mathcal{X}$ and an ordering $\pi_d(A)$, let $Q_d\left(A, \pi_d(A)\right)$ denote the optimal*

score of a network with maximum in-degree d on A, which is consistent with $\pi_d(A)$:

$$Q_d\left(A, \pi_d(A)\right) = \sum_{X_j \in A} F_d\left(X_j, \{X_q \mid X_q \text{ precedes } X_j \text{ in } \pi_d\left(A\right)\}\right).$$

Optimal score of a bounded node in-degree network on $A \subseteq \mathcal{X}$ is then given by $Q_d^*(A) = Q_d\left(A, \pi_d^*\left(A\right)\right)$. As before, function Q_d can be computed using a recursive formulation as follows:

$$X_j^* = \underset{X_j \in A}{\operatorname{argmax}}\, Q_d\left(A, \pi_d^*(A - \{X_j\})X_j\right)$$

$$= \underset{X_j \in A}{\operatorname{argmax}}\, \left(F_d\left(X_j, A - \{X_j\}\right) + Q_d^*\left(A - \{X_j\}\right)\right),$$

$$Q_d^*\left(A\right) = F_d\left(X_j^*, A - \left\{X_j^*\right\}\right) + Q_d^*\left(A - \left\{X_j^*\right\}\right).$$

Using the above definitions of F_d and Q_d in place of F and Q, the dynamic programming algorithm described in Section 2 works in a similar fashion. Note that the algorithm that places no restriction on in-degree can be thought of as a special case of the bounded in-degree algorithm for $d = n - 1$.

3.1.2 Characterizing Run-Time Complexity as a Function of d

Intuitively, the run-time complexity of structure learning should increase with increase in d. We again consider the class of scoring functions which can be computed in linear time on the underlying data to be considered – i.e., computing $s(X_i, A)$ takes $O(m|A|)$ time. Consider $d \leq \lfloor \frac{n}{2} \rfloor$. In computing functions F_d and Q_d over all nodes in the lattice, the s functions are computed only on subsets of size $\leq d$. Therefore, the run-time complexity is given by

$$O\left(\sum_{i=0}^{n} \binom{n}{i}[(n-i)\cdot i + i] + \sum_{i=0}^{d} \binom{n}{i}[(n-i)\cdot m \cdot i + i]\right) = O\left(n^2 \cdot 2^n + m \cdot n \cdot d^2 \cdot \binom{n}{d}\right).$$

Lemma 3.1 *If* $d < \frac{1}{3}n - 2\log n$, *then* $d^2\binom{n}{d} = O\left(2^n\right)$.

Proof 2 $\binom{n}{i+1} = \frac{n-i}{i+1} \cdot \binom{n}{i}$. *Therefore,* $\binom{n}{i+1} \geq 2\binom{n}{i}$ *if and only if* $\frac{n-i}{i+1} \geq 2$, *or* $i \leq \frac{1}{3}n - \frac{2}{3}$. *Let* $d < \frac{1}{3}n - 2\log n$ *and* $j = 2\log n - 1$. *Consider the* $j+1$ *consecutive binomial coefficients* $\binom{n}{i}$ *for* $d \leq i \leq d+j$. *Each successive binomial coefficient is at least twice as big as the previous one. Then:*

$2^n = \sum_{i=0}^{n} \binom{n}{i}$

$> \sum_{i=d}^{d+j} \binom{n}{i} + \sum_{i=n-d-j}^{n-d} \binom{n}{i}$

$= 2 \cdot \sum_{i=d}^{d+j} \binom{n}{i}$

$\geq 2 \cdot \left(1 + 2 + 2^2 + \cdots + 2^j\right) \cdot \binom{n}{d}$

$> 2 \cdot 2^j \cdot \binom{n}{d}$

$= n^2 \cdot \binom{n}{d}$

$> d^2 \cdot \binom{n}{d}$.

Corollary 3.2 *For* $d < \frac{1}{3}n - \log mn$, *the run-time complexity of structure learning is* $O(n^2 \cdot 2^n)$.

Proof 3 *For* $d < \frac{1}{3}n - 2\log n - \log\frac{m}{n} = \frac{1}{3}n - \log mn$, *an additional* $\log\frac{m}{n}$ *consecutive binomial coefficients exhibit the doubling property. Thus,* $\frac{m}{n} \cdot d^2 \cdot \binom{n}{d} = O(2^n)$.

Corollary 3.2 is significant because it stipulates that for d values up to the bound prescribed, the asymptotic run-time complexity is as if no s functions are computed at all. Thus, there is no value to choosing d below this bound and such a choice would be needlessly restrictive without any gains in the run-time. Although networks for large n cannot be computed in parallel due to exponential dependence of run-time on n, it is interesting to note that d can asymptotically grow as $\frac{1}{3}n$ without impacting the run-time complexity. If we can reasonably assume that no

node will have more than as many parents, structure learning becomes unaffected by the cost of computing s functions. Note that Lemma 3.1 does not provide a tight bound as it ignores the contribution of all but $2(j+1)$ terms in the $n+1$ binomial coefficient summation. As a result, the value below which decreasing d does not provide any tangible performance gains is likely to be even higher in practice. This will be empirically shown in the next Section.

Lemma 3.3 *For $d \geq \lceil \frac{n}{2} \rceil$, the asymptotic run-time complexity is the same as for $d = n-1$.*

Proof 4 *Follows from a straightforward inspection of the run-time, which is*

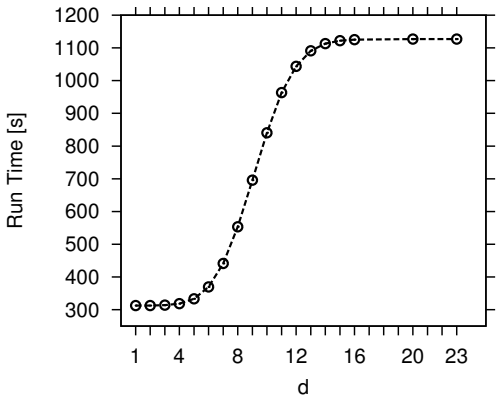
$$O\left(\sum_{i=0}^{n} \binom{n}{i} ((n-i)\cdot i + i) + \sum_{i=0}^{d} \binom{n}{i} ((n-i)\cdot m \cdot i + i)\right).$$

In the second term, given the symmetry of binomial coefficients, the summation for $d \geq \lceil \frac{n}{2} \rceil$ is at least half as much as the summation for $d = n-1$.

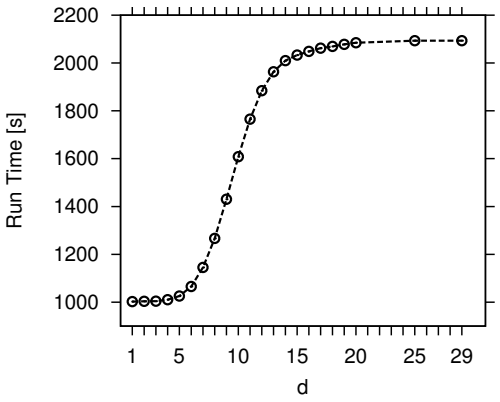
Taken together, Corollary 3.2 and Lemma 3.3 establish a range of values of d where run-time performance varies as a function of d. There is no reason to choose a value of d smaller than what is established by Corollary 3.2 as it restricts the in-degree further without providing any compensatory gains in run-time. Similarly, choosing a value of d greater than or equal to $\lceil \frac{n}{2} \rceil$ is unnecessary as the run-time complexity is equivalent to the unrestricted case.

3.2 Experimental Analysis for Bounded Node In-degree

In this set of experiments, we investigated the empirical behavior of the modified parallel algorithm described in Section 3.1. For a given test data with n genes and m observations, we recorded the run-time for learning the BN structure while varying d from 1 to $n-1$. We considered two data sets of $n = 24$ and $n = 30$ genes, respectively, both with $m = 200$



(a) Number of genes: 24, number of cores: 32



(b) Number of genes: 30, number of cores: 2,048

Figure 3.1 Run time as a function of the node in-degree bound d of the modified parallel algorithm for BN structure learning. Test data for (a) $n = 24$ genes executed on 32 cores, and (b) $n = 30$ genes executed on $2,048$ cores, for varying $d = 1, 2, \ldots, n-1$.

observations. The experiments were run on the IBM Blue Gene/P system. The run-times, in seconds, as a function of d are summarized in Figure 3.1.

Based on previous analysis, we expect the run-time to be flat until d crosses the bound specified by Corollary 3.2, raising as a function of d until reaching a second plateau after d reaches $\frac{1}{2}n$. This behavior can be clearly seen in Figures 3.1(a) and 3.1(b), for two different values of n when run on two different numbers of cores, respectively. The onset of the second plateau is around the $d = \frac{1}{2}n$ point in both cases, closely conforming to the theoretical prediction. Note that since $m = 200$ and n is relatively small, the bound predicted by Corollary 3.2 in both cases actually turns out to be negative. However, as pointed out, this bound is rather conservative due to the relaxed analysis and the value of d at which the run-time begins its upward trajectory is expected to be better than the theoretical prediction. In case of the experiments shown, this happens at $d = 4$ for $n = 24$ and at $d = 5$ for $n = 30$.

3.3 Contributions

To our knowledge, the bounds established for run-time as a function of the maximum in-degree are new both in sequential and parallel settings.

CHAPTER 4. Parallel Algorithmic Framework for Large-scale Bayesian Network Structure Learning

4.1 Introduction

To enable large-scale BN inference heuristic-based serial algorithms have been developed, which are still not suitable for gene network modeling on a whole genome scale for large genomes [Tsamardinos et al., 2006]. To extend heuristic-based BN inference to networks of thousands of variables and observations, we develop a two-phase parallel approach. In phase one, a set of candidate parents is learned for each variable, utilizing a state of the art constraint-based heuristic method. In phase two, a heuristic score-optimization method is developed, which is inspired by exact parents learning algorithm, to induce the BN structure from the pre-computed in phase one sets of candidate parents. The two phases are independent in the sense that phase two admits any other method for the pre-computation of candidate parents. In addition, we extend phase one to learning Markov boundaries. Performance of each phase is assessed with respect to achieved speedup and ability to solve larger problems, as well as, quality of the learned network on synthetic data.

4.1.0.1 Previous Work

As discussed in Chapter 2, exact structure learning of BNs is NP-hard even for the instance of the problem when the node in-degree is bounded [Chickering et al., 1994]. Exact approaches

explore the super-exponential search space of all possible directed acyclic graphs (DAG) over a given set of variables [Robinson, 1973]. While in recent works [Ott et al.] the run-time has been reduced to exponential, the memory requirement also grows exponentially in the number of variables. This is often a serious bottleneck, even for modest problem sizes involving few tens of variables. This has prompted the development of heuristic approaches, which remains an active area of research [Chen et al., 2006, Cheng, 2002, Chickering, 2002, Friedman et al., Heckerman et al., 1995a, Moore and Wong, 2003, Pe'er et al., 2006, Spirtes et al., 2001, Spirtes and Meek, 1995, Tsamardinos et al., 2006, Wang et al., 2007].

Most heuristic approaches for BN structure learning can be characterized as either constraint-based, or based on score optimization, or hybrid approaches that include aspects of both. In constraint-based approaches, conditional independencies between variables are evaluated using statistical tests (G^2, partial correlation, etc.) and propagated, using a graphical criterion called d-separation, to infer a partially oriented network. Then, axioms are used to orient the remaining edges such that all constraints are satisfied [Spirtes et al., 2001, Cheng, 2002]. These methods have produced mixed results, due largely to the sensitivity of the statistical tests to noise in the data.

Score optimization methods optimize a scoring criterion while exploring the search space [Friedman et al., Moore and Wong, 2003, Cooper and Herskovits, 1992, Teyssier and Koller, 2005]. The scoring criterion evaluates how well a proposed graph predicts dependencies in the observed data. Commonly, a classic hill-climbing technique over the search-space of DAGs is employed. At each step, every possible edge addition, removal, and reversal is evaluated under the acyclicity constraint, and a step is made towards the highest scoring change. These steps are repeated until the overall network score can no longer be improved. This classic idea has

been enhanced by adding TABU lists and random restarts to avoid local optima. A novel operator has also been proposed, which allows for the reinsertion of a particular node in a new location in the graph [Moore and Wong, 2003]. Variations of this stream of approaches over DAGs have been proposed, where considerations over orders are made [Cooper and Herskovits, 1992, Teyssier and Koller, 2005]. These methods explore reductions in search space imposed by a topological sort on a DAG. Alternatively, relaxing the acyclicity constraint has been proposed, which allows for the computation of optimal set of neighboring nodes (both incoming and outgoing edges). Then, edges are removed to break cycles and to eventually restore the acyclicity constraint, such that the overall network score is optimized [Peng and Ding, 2003]. Reductions in search space have been proposed by pre-selecting subsets of variables for each node. One way to do this is to use constraint-based approach and eliminate edges between conditionally independent variables. Such methods are sometimes referred to as hybrid methods [Tsamardinos et al., 2006].

An extensive evaluation of the performance of several state-of-the-art algorithms, representing each of the three categories described above, including the *Sparse Candidate* [Friedman et al.], *Three Phase Dependency Analysis* [Cheng, 2002], *Optimal Reinsertion* [Moore and Wong, 2003], *Greedy Equivalence Search* [Chickering, 2002], *PC* [Spirtes et al., 2001], and *Max-Min Hill Climbing (MMHC)* [Tsamardinos et al., 2006] algorithms, highlighted $MMHC$ as the most accurate approach and scalable to large datasets [Tsamardinos et al., 2006]. Even though the presented method is relatively fast in determining the skeleton network, it is reported to take about 19 hours for a domain of approximately $5,000$ variables and an additional 13 days to orient its edges [Tsamardinos et al., 2006]. This category of approaches is generally considered a promising direction.

MMHC uses heuristic procedures to pre-compute the set of parents and children (PC) of each variable. Then, it optimizes the overall network score by selecting parents for each variable from its respective PC, using hill-climbing.

In stark contrast to the vast literature on serial algorithms for BN structure learning, and despite the compute and memory intensive nature of the problem, there is limited work in the development of parallel methods for structure learning. In 2009, Nikolova *et al.* [Nikolova et al., 2009] developed the first parallel algorithm for exact BN structure learning that is both work and space optimal. While this work allows pushing the limits on the scale of networks that can be inferred with precision, the NP-hard nature of the problem limits the applicability of this solution to small networks of 30-40 nodes. Heuristics based approaches that trade off optimality for the ability to learn larger networks are the only means to structure learning of large-scale BNs. While many score optimization and hybrid heuristic approaches that yield good quality results have been reported, there is little work in developing efficient parallel counterparts to these algorithms. Some works have resorted to simplistic parallelization of such approaches which may adversely affect quality. For example, Tamada *et al.* [Tamada et al., 2011] build whole-genome scale networks by selecting a random subset of nodes on which a serial method can be run, and exploiting parallelism by conducting many such runs with multiple random subsets. The results of these runs are subsequently combined to form the larger network. While the quality of the network improves with the number of random samples, the true parents of any given node may not be simultaneously part of any one of the subsets considered.

4.2 Phase 1: Parallel Discovery of Direct Causal Relations and Markov Boundaries with Applications to Gene Networks

In this section, we present parallel algorithms for two problems that arise in relation with network structure learning and analysis: (i) the discovery of candidate parents for each variable (the set of parents and children of each node in the corresponding Bayesian network), and (ii) the computation of Markov boundary of each variable, defined as the minimal set of variables that shield the target variable from all other variables in the domain. Our parallel algorithms are based on state-of-the art constraint-based heuristic optimization methods. They are shown to be work-optimal and communication efficient, and exhibit nearly perfect scaling.

4.2.0.2 Contributions

Our goal is to push BN structure learning towards ever larger domains to facilitate applications such as whole-genome network inference. We present parallel algorithms for direct causal relations inference, which corresponds to the discovery of the parents and children (PC) nodes for each variable in the network.

We then consider the problem of inferring Markov boundaries (MB). The MB of a given variable is defined as a minimal set of nodes which shield the particular variable from all other variables in the domain. While computing MBs by itself is of great interest in bioinformatics, as it has various implications for feature selection in network analysis, knowing MBs in addition to PCs introduces new constraints which can reduce the search space in the optimization step (for edge orientation) towards the full BN discovery. A recent study reported over 2 hours time for determining a Markov boundary of a single variable over a domain of 139,351 variables [Pena et al., 2007]. At this rate, computing the Markov boundaries for all variables would take

an estimated 32 years.

4.2.1 Direct Causal Relations (PC) and Markov Boundaries (MB) Learning

We next review two state-of-the-art serial algorithms for PC and MB discovery, which form the basis of our parallelization. The first algorithm, named $MMPC$ (Max-Min Parents and Children) [Tsamardinos et al., 2006], identifies the set of parents and children for a given target variable. The second algorithm is our modification of the $PCMB$ (Parents and Children based Markov Boundary algorithm) [Pena et al., 2007]. We refer to the modified $PCMB$ algorithm as $PCMB^*$. Under the assumptions that the set of observations D is an independent and identically distributed sample from the probability distribution P and that the tests of conditional independence and measures of conditional dependence are correct, the outputs of $MMPC$ and $PCMB$ are $PC(T)$ and $MB(T)$ for the DAG faithful to P (proofs in [Tsamardinos et al., 2006] and [Pena et al., 2007], respectively).

1. {Phase 1: Forward}
2. **CPC** $\leftarrow \emptyset$
3. **repeat**
4. $\quad \langle F, assocF \rangle = MaxMinHeuristic(T; \mathbf{CPC})$
5. $\quad$ **if** $assocF \neq 0$ **then**
6. $\quad\quad \mathbf{CPC} = \mathbf{CPC} \cup F$
7. $\quad$ **end if**
8. **until** **CPC** does not change
9. {Phase 2: Backward}
10. **for all** $X \in \mathbf{CPC}$ **do**
11. $\quad$ **if** $X \perp\!\!\!\perp T | \mathbf{Z}$ for some $\mathbf{Z} \subseteq \mathbf{CPC}$ **then**
12. $\quad\quad \mathbf{CPC} = \mathbf{CPC} \backslash \{X\}$
13. $\quad$ **end if**
14. **end for**
15. **return** **CPC**

Algorithm 1: $\overline{MMPC}(T, D)$

1. Procedure $MaxMinHeuristic(T, \mathbf{CPC})$
2. $assocF = \max_{X \in \mathcal{X} \setminus (\mathbf{CPC} \cup \{T\})} MinAssoc(X; T|\mathbf{CPC})$
3. $F = \operatorname{argmax}_{X \in \mathcal{X}} MinAssoc(X; T|\mathbf{CPC})$
4. **return** $\langle F, assocF \rangle$

1. $\mathbf{CPC} = \overline{MMPC}(T, D)$
2. **for all** $X \in \mathbf{CPC}$ **do**
3. **if** $T \notin \overline{MMPC}(X, D)$ **then**
4. $\mathbf{CPC} = \mathbf{CPC} \setminus \{X\}$
5. **end if**
6. **end for**
7. **return CPC**

Algorithm 2: $MMPC(T, D)$

4.2.1.1 $MMPC$

$MMPC$ receives a target variable T and a set of observations D as input and returns $PC(T)$. It proceeds in two stages as outlined in Algorithms 1 and 2. First, procedure $\overline{MMPC}$ identifies a superset of PC which may potentially contain false positives. In phase 1, $\overline{MMPC}$ repeatedly uses the $MaxMinHeuristic$ to continually add the next most probable variable to include in the candidate PC set called CPC. In phase 2, $\overline{MMPC}$ conditions on all possible subsets of CPC to remove some false positives which may have been added to CPC in phase 1. Finally, $MMPC$ iterates over the results of $\overline{MMPC}$ enforcing the symmetry properties of the PC set, to further eliminate false positives that could occur due to conditioning only on subsets of CPC. The $MaxMinHeuristic$ uses procedure $MinAssoc$ to compute the minimum association between two variables T and X given a subset $\mathbf{Z}$, which is defined as the minimum association between the two variables conditioned on every possible subset of $\mathbf{Z}$:

Definition 4.1

$$MinAssoc(X; T|\boldsymbol{Z}) = \min_{\boldsymbol{S} \subseteq \boldsymbol{Z}} Assoc(X; T|\boldsymbol{S}).$$

Time complexity is estimated in the number of computed independence tests. In the worst case, phases 1 and 2 of $\overline{MMPC}$ are bounded by $O(|\mathcal{X}| \cdot 2^{|CPC|})$ and $O(|CPC| \cdot 2^{|CPC|-1})$, respectively. Due to sample size limitations in statistically reliable independence testing with real data, as described in [Spirtes et al., 2001], conditioning is only possible on subsets of size up to l, which results in time complexity $O(|\mathcal{X}| \cdot |CPC|^{l+1})$. Furthermore, the sizes of the larger CPC sets found at the end of phase 1 by $MMPC$ are claimed to be of the same order as the sizes of the corresponding final $CPC = PC$ sets produced by the algorithm. With this expectation, the total computational complexity of running $MMPC$ for all variables in $\mathcal{X}$ is estimated as $O(|\mathcal{X}|^2 \cdot |PC|^{l+1})$, where PC is the largest set of parents and children over all variables in $\mathcal{X}$.

4.2.1.2 $PCMB^*$

Once the PCs are identified, we proceed to find the MBs. The $PCMB$ algorithm in [Pena et al., 2007] utilizes a different procedure for first learning PC sets. We replace this procedure with $MMPC$ and arrive at the modified $PCMB^*$ algorithm outlined in Algorithm 3. $PCMB^*$ receives a target variable T and a set of observations D as input and returns $MB(T)$. It first sets $MB(T)$ to $PC(T)$ as returned by $MMPC$. Then, the remaining parents of T's children (the so called spouses) are added. Since $MMPC$ would have already identified the set $\mathbf{Z}$ that d-separates X and T at line 6 in Algorithm 3, the worst case complexity of $PCMB^*$ over all variables in $\mathcal{X}$ is $O(|\mathcal{X}| \cdot |PC|^2)$.

4.2.1.3 Conditional Independence Tests and Association Measures

To test whether $X_i \perp\!\!\!\perp X_j | \mathbf{X_k}$, and to assess the strength of association between X_i and X_j conditioned on $\mathbf{X_k}$ when independence does not hold, the G^2 statistic is used [Spirtes et al.,

1. **MB** = **PC** = $MMPC(T, D)$
2. **for all** $Y \in$ **PC do**
3. **for all** $X \in MMPC(Y, D)$ **do**
4. **if** $X \notin$ **PC then**
5. find $\mathbf{Z} : X \perp\!\!\!\perp T|\mathbf{Z}$ and $X, T \notin \mathbf{Z}$
6. **if** $X \perp\!\!\!\perp T|\mathbf{Z} \cup \{Y\}$ **then**
7. **MB** = **MB** $\cup \{X\}$
8. **end if**
9. **end if**
10. **end for**
11. **end for**
12. **return MB**

Algorithm 3: $PCMB^*(T, D)$

2001]. Adopting the notation from Tsamardinos *et al.*, [Tsamardinos et al., 2006], let $S_{ijk}^{ab\mathbf{c}}$ be the count of occurrences in the data where $X_i = a$, $X_j = b$, and $\mathbf{X_k} = \mathbf{c}$. Then

$$G^2 = 2 \sum_{a,b,c} S_{ijk}^{ab\mathbf{c}} \cdot \ln \frac{S_{ijk}^{ab\mathbf{c}} S_k^{\mathbf{c}}}{S_{ik}^{a\mathbf{c}} S_{jk}^{b\mathbf{c}}},$$

with corresponding degrees of freedom given by

$$(r_i - 1) \cdot (r_j - 1) \cdot \prod_{X_l \in \mathbf{X_k}} r_l,$$

where r_i is the size of the domain (number of distinct values) of variable X_i. The G^2 statistic is computed under the null hypothesis of independence. The G^2 statistic is asymptotically distributed as χ^2. If the null hypothesis is rejected, which is interpreted as $X_i \not\perp\!\!\!\perp X_j|\mathbf{X_k}$, then the negative p-value of the test is used as an approximation of the measure of association between X_i and X_j conditioned on $\mathbf{X_k}$. Otherwise, the variables X_i and X_j are considered conditionally independent, which is equivalent to an association measure of 0.

In the following two sections, we present parallel algorithms for computing causal relations and Markov boundaries. While based on the sequential algorithms reviewed in the previous

section, we do not always directly adhere to the specific flow of these algorithms as it would be counterproductive to efficient parallelization. However, the algorithms are logically equivalent and produce identical results.

4.2.2 Parallel Direct Causal Relations (PC) Discovery

4.2.2.1 Algorithm

The sequential algorithm calls $MMPC$ for each variable in $|\mathcal{X}|$, the first step of which is a call to $\overline{MMPC}$. In the parallel algorithm, we first compute $\overline{MMPC}$ for all the variables collectively in parallel. This component is straightforward. Initially, all $|\mathcal{X}|$ variables are evenly distributed across p processors. Each processor is responsible for computing the CPCs of its designated variables. As mentioned previously, in phase 1 these sets are built up incrementally by adding one element at a time, while in phase 2 some of the false positives are removed. Note that the CPC set sizes are not uniform across all variables either at the end of phase 1 or at the end of phase 2. However, we found that an equal allocation of variables to processors is sufficient to ensure good load balancing at this stage. In addition, we suggest a way to reuse computations in phase 1 for speedy execution of phase 2, which is possible if a variable is allocated to the same processor for both phases.

In order to compute the $MaxMinHeuristic$ efficiently, we restate Definition 1 in a Dynamic Programming (DP) formulation, i.e.:

$$MinAssoc(X;T|\mathbf{Z}) = \min \begin{cases} Assoc(X;T|\mathbf{Z}) \\ \min\limits_{\forall Y \in \mathbf{Z}} MinAssoc(X;T|\mathbf{Z}\backslash\{Y\}) \end{cases}$$

In phase 1, all processors first compute their expanded CPC's for each of their local variables. We save previous calls to $MinAssoc$ in memory, and reuse them as needed in phase 2. The DP formulation and preserved locality allow for more efficient independence testing in both phases 1 and 2. To check if any subset $\mathbf{Z} \subseteq \mathbf{CPC}$ exists that d-separates X and T (line 11 in algorithm 1), it is sufficient to determine if $MinAssoc(X; T|\mathbf{CPC}) = 0$. Thus, if any of the subsequent calls to $MinAssoc$ conditioned on subsets of CPC have already been computed, these results are reused.

The remaining part of $MMPC$ is to perform symmetry testing: $X \in CPC(Y)$ is removed from $CPC(Y)$ whenever $Y \notin CPC(X)$. The sequential algorithm performs this test for each CPC set, and for each variable within that set. We perform this set reduction simultaneously in parallel on all sets as follows. We first transform each CPC set to a doubly redundant list of pairs representation. Let Y denote a variable assigned to a processor, and let $X \in CPC(Y)$. We form two pairs (X, Y) and (Y, X). Thus on each processor we form the list consisting of all such pairs generated for all of its local $|\mathcal{X}|/p$ variables and their respective non-empty CPC sets. A parallel sort is then used to sort all the pairs so generated using the first variable in the pair as primary key and the second variable as the secondary key.

If $X \in PC(Y)$ then the pairs (X, Y) and (Y, X) would have been generated on the processor to which X was originally distributed. Analogously, if $Y \in PC(X)$ then the same pairs would have been generated on Y's processor. After sorting, both sets of duplicates will appear adjacent. If $Y \notin PC(X)$, then the pairs (X, Y) and (Y, X) will occur only once in the sorted list. Each processor scans its local portion of the globally sorted list of pairs to remove pairs which occur exactly once, and retain a single copy of those pairs which occur twice. This duplicate check effectively implements lines 3-4 in algorithm $MMPC$. The special case of two

consecutive duplicated pairs split across neighboring processors can be easily taken care of by communicating the first element of each processor to the processor with previous rank. At the end of this stage, we have a list of pairs representation of the PC set of each variable.

4.2.2.2 Run-Time Complexity

Aside from the initial computations that each processor performs on its designated block of data of size $|\mathcal{X}|/p$, the parallel PC algorithm is based on parallel sort. Initially each processor receives $|\mathcal{X}|/p$ variables and in $\overline{MMPC}$ proceeds to compute their CPCs. As made in the serial algorithm and verified experimentally, we make the assumption that the work for determining any CPC set is of the same order as the largest PC set. The complexity of parallel execution of $\overline{MMPC}$ is bounded by the run-time per processor for phases 1 and 2, i.e., $O\left(\frac{|\mathcal{X}|\cdot|PC|^{l+1}}{p}\right)$, which is work-optimal. In $MMPC$ a total of $2\cdot\frac{|\mathcal{X}|}{p}\cdot|PC|$ mirror CPC-pairs are generated on each processor and parallel sort is invoked on a total number $2|\mathcal{X}|\cdot|PC|$ pairs. This concludes the PC discovery for all variables. The total run-time is expected to be dominated by the first phase due to exponential dependence on l.

4.2.3 Parallel Markov Boundaries (MB) Discovery

4.2.3.1 Algorithm

The sequential algorithm calls $PCMB^*$ for each variable T in $\mathcal{X}$. It includes $PC(T)$ and proceeds to examine all candidate spouses X of T, i.e. $X \in PC(Y)$ for $Y \in PC(T)$. It then retrieves the set $\mathbf{Z}$ which was previously found by a call to $MinAssoc$, and therefore has already been recorded, to test for independence between the candidate spouse X and T given the union of set $\mathbf{Z}$ and the hypothetical child Y of X and T. Sequentially, the candidate spouses are determined in two nested for-loops (lines 2-4). We find all candidate spouses X

of all variables T in $\mathcal{X}$ simultaneously in parallel as follows.

We continue to utilize the pair-list representation introduced in section IV and consider the output of the parallel PC procedure. Each processor now stores a sorted list of PC pairs, where a pair (X, Y) is present if $X \in PC(Y)$ and $Y \in PC(X)$. Implementing the candidate spouse check in pair-representation of PC sets is equivalent to checking whether, for any two pairs (T, Y) and (Y, X) in the list, the pair (X, T) is also present. Once all such *spouse* pairs are identified, they are returned to the processors which would have already computed their respective set $\mathbf{Z}$ for the final independence test. We describe this procedure in detail in five steps below.

Spouse-pairs Generation Each processor parses its local list of pairs to find all pairs with the same primary key. The lists are sorted and therefore all such pairs occur consecutively. For any two pairs with the same primary key a third pair is generated from the remaining two non-shared variables. This third pair is in effect a candidate spouse-pair. The order of the variables within any spouse-pair is irrelevant. Recall that for any two variables X and Y s.t. $X \in PC(Y)$ both pairs (X, Y) and (Y, X) were generated by the PC procedure. This ensures that checking only the primary key for candidate-spouse-pair generation is sufficient. The reason for doing so is to ensure that all pairs which can potentially be used to generate spouse-pair occur consecutively to maximize locality. Parsing proceeds starting with the first pair on the list and comparing it to each of the following pairs until the primary keys are no longer the same. In some cases, a sequence of pairs with a common primary key can be split between processors, i.e. some pairs may have a common primary key with the last pair on a given processor. In those cases, all such pairs are communicated to the neighbor processor of higher rank, in order to continue the process of candidate spouse-pair generation. One last

detail of this procedure is to record the primary key of the two generating pairs for each newly created candidate spouse-pair. We refer to this key as *generator* and we will return to it in the last step of this algorithm.

Mixing of Spouse-pairs and PC Pairs and Re-sorting Once all possible spouse-pairs are created and their corresponding *generators* recorded, these pairs are marked and mixed with the original PC pairs. A global parallel sort is invoked again to re-sort the new list of all pairs.

Search and Retention of Singly-occurring Spouse-pairs After sorting, duplicate pairs occur adjacently. The presence of two duplicate pairs implies that the generated candidate spouse-pair is already present in the PC list, i.e., that $X \in PC(T)$ (line 4 in $PCMB^*$). We are interested in the case when $X \notin PC(T)$, and therefore are looking for pairs which occur only once. Once all such pairs are identified and retained, others are removed from the list.

Distribution of Retained Spouse-pairs to Designated Processors Proceeding with the final independence testing requires that each candidate spouse-pair be delivered to the processor which had originally d-separated X and T and thus computed the set $\mathbf{Z}$. Clearly, this was done by the processors to which variables X and T were originally distributed. Thus, it suffices to deliver each spouse-pair to one of these two processors.

Computation of Markov Boundaries Once each processor receives its designated spouse-pairs, procedure *Compute Markov Boundaries* outlined in Algorithm 4 takes place. This procedure returns partial MBs for each spouse-pair. Since MBs are also symmetrical [Neapolitan, 2003], namely if $X \in MB(Y)$ then $Y \in MB(X)$, the partial $MB(X,T)$ is recorded for the pair. For each variable T, complete $MB(T)$ is acquired by taking the union of all such partial MBs where the variable T is present, in addition to $PC(T)$. Recall that in this setting the spouse-pair (X,T) is such that there exists $Y : Y \in PC(T), X \in PC(Y)$, while Y is also

the generator variable. Once set $\mathbf{Z}$ is obtained, pair (X, T) is added to the partial MB list if $Assoc(X; T|\mathbf{Z} \cup Y)$ returns 0.

1. $MP \leftarrow$ {received spouse-pairs}
2. **for all** $(T, X) \in MP$ **do**
3. find $\mathbf{Z} : T \perp\!\!\!\perp X|\mathbf{Z}$ and $T, X \notin \mathbf{Z}$
4. **if** $T \perp\!\!\!\perp X|\mathbf{Z} \cup \{generator(T, X)\}$ **then**
5. $MB_{T,X} = MB_{T,X} \cup \{generator(T, X)\}$
6. **end if**
7. **end for**

Algorithm 4: Compute Markov Boundaries

4.2.3.2 Run-time Complexity

Spouse-pairs Generation In order to generate spouse-pairs all mirror PC pairs are examined. Each variable X participates in creating exactly $2 \cdot |PC(X)|$ mirror PC pairs. It can therefore be a *generator* for spouse-pairs at most $\binom{|PC(X)|}{2}$ times. Having mirror PC pairs does not double the run-time of this stage. This is because spouse-pairs are only generated when the first variable of the two pairs which are being compared are the same. Hence, a given variable could only serve as a *generator* when it is in the first position of a PC mirror pair. This results in overall compute work of $O\left(\binom{|PC|}{2} \cdot \frac{|\mathcal{X}|}{p}\right)$. In boundary cases when some pairs need to be sent to the adjacent processor the cost of communication is added but negligible. This communication cannot be overlapped with computation because it is followed by a global sorting procedure on all spouse-pairs so generated and previously recorded PC pairs.

Mixing of Spouse-pairs and PC Pairs and Re-sorting After mixing the mirror PC pairs and all generated spouse-pairs, the number of pairs per variable is bounded by $q' = 2|PC| + \binom{|PC|}{2}$. These pairs are sorted using a parallel sort.

Search and Retention of Singly-occurring Spouse-pairs Checking for duplicates requires the local list of sorted pairs on each processor to be scanned once, which requires linear time

in the size of the local list. Under the assumption that the resulting local lists of sorted pairs are of approximately equivalent size, this operation is bounded by $O\left(q' \cdot \frac{|\mathcal{X}|}{p}\right)$.

Distribution of Retained Spouse-pairs to Designated Processors An All-to-all communication is used to deliver the lists of retained spouse-pairs to their respective destinations. In terms of complexity, the parallel sorting needed for many of the steps of the algorithm dominates this step (as all-to-all communication is a lower bound for parallel sorting).

Computation of Markov Boundaries In this stage each processor retrieves one locally stored result of a call to $MinAssoc$ and makes at most one call to $Assoc$. Since complexity of the call to the independence test function is taken as constant, this operation is linear in the size of the local list of retained spouse-pairs. The sizes of these lists are difficult to compute precisely, but under the assumption that each processor receives approximately the same number of spouse-pairs, the cost of this step is balanced.

Overall, the parallel algorithm for computation of all Markov boundaries relies on parallel sorting as the dominant component of the run-time. The largest size of the data sorted is $O(|\mathcal{X}| \cdot |PC|^2)$. Our implementation relies on Sample sort.

4.2.4 Experimental Results

The presented algorithms were implemented in C++ and MPI. To assess the performance and applicability of the algorithms experiments were performed on up to 255 16-way SMP compute nodes, totaling 4,080 AMD Opteron processors of the TACC Ranger system (core frequency of 2.3 GHz and 32 GB of memory per node). As described in the previous section, parallel sort is the most communication-intensive operation used in the algorithms. The remaining work is the dominant part, consisting of well balanced local computations with negligible communication at the boundaries. Therefore, one would expect good scaling. For

the purpose of demonstrating scalability we first present results from two experiments on medium-size datasets, where we analyze how the number of input variables and sample size affect performance. Then we demonstrate the applicability of our methods to real-world problems by learning the PCs and MBs for a whole-genome unoriented BN of the model plant *Arabidopsis thaliana*.

Following the practice used in Spirtes *et al.*, Tsamardinos *et al.*, and Pena *et al.*, we implement the G^2 statistic as a measure for association strength. We adopt the same sample size requirements for reliable independence test assessment. That is, for any independence test to be considered reliable, at least five observations on average per estimated parameter are required [Spirtes et al., 2001]. The assessment of the correctness of the inferred results is beyond the scope of this paper, as it varies with respect to the independence testing criteria, assumptions about statistical tests reliability and other parameters.

4.2.4.1 Data Preparation

A total of $3,546$ non-redundant Affymetrix expression profiles for *Arabidopsis thaliana* were collected from the NASCArray [NAS, 2009], AtGenExpress [TAI, 2009], ArrayExpress [Arr, 2009], and GEO [GEO, 2009] repositories and pre-processed as described in [Zola et al., 2010]. This resulted in a data-matrix of $3,137$ observations and $15,596$ probe-sets, mapping to $15,495$ genes. We refer to this dataset as *whole-genome* dataset. In addition, to assess the performance and scalability of the presented parallel algorithms we created three medium-scale datasets, each consisting of $1,000$ genes and 500 observations, $3,000$ genes and 250 observations, and $3,000$ genes and 500 observations, respectively. These datasets were obtained by selecting subsets and subsamples of the *whole-genome* dataset.

The values for each variable are the gene expression levels detected by microarray exper-

Table 4.1 Total run-time for the computation of $PC + MB$ for the dataset with $m = 500$ observations and varying number of genes denoted by n.

No. of	Time [s]	
CPU cores	$n = 1,000$	$n = 3,000$
16	173	3,105
32	90	1,594
64	46	805
128	24	413
256	15	209
512	9	108

iments, which are continuous. We represent gene expression using a multinomial model, i.e. we treat each variable as discrete and learn a multinomial distribution describing the probability of each state of a given variable given the state of its parents. After the normalization performed in [Zola et al., 2010], we discretized the data using the average expression value of each gene across experiments as control in a similar manner as in [Friedman et al., 2000]. The multinomial model allows for the detection of combinatorial effects, e.g. that a given gene may be expressed only if a set of other genes are jointly expressed, as opposed to not being expressed if even one of the genes in this set is not expressed.

4.2.4.2 Performance Analysis for Varying Number of Variables

In the first experiment we analyzed the performance of the algorithms with respect to the number of input variables. We tested the algorithms on the datasets with $n = 1,000$ and $n = 3,000$ genes, and a fixed $m = 500$ number of observations. The run-time as a function of the number of processor cores used are shown in Table 4.1. The same data is plotted in Figure 4.1 to show relative speedup when compared to a 16-core run. These results were generated using the standard significance level cutoff of 0.001.

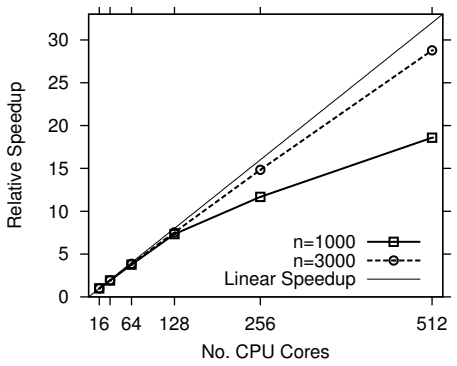


Figure 4.1 Relative speedup when compared to a 16-core run for the dataset with $m = 500$ observations and varying number of genes denoted by n.

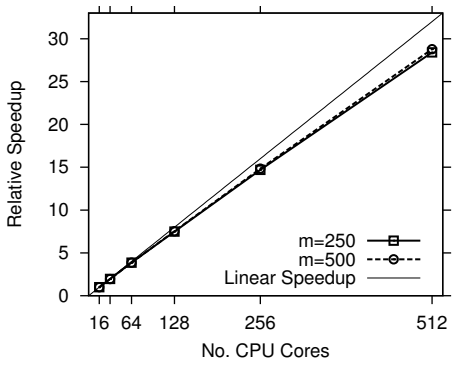


Figure 4.2 Relative speedup when compared to a 16-core run for the dataset with $n = 3,000$ genes and varying number of observations denoted by m.

As can be seen, for the larger dataset ($n = 3,000$) linear scaling is observed up to 512 cores, while in the case of the smaller dataset ($n = 1,000$) the performance gains are tempered for more than 128 cores. This can be explained by the much smaller grain-size at 512 cores, which in this case translates to a meager 1-2 variables per processor. Thus, the parallel algorithm works efficiently even for a rather small grain-size of 5-6 variables per processor. For large problems involving over 100,000 variables, this permits utilization of over ten thousand processors.

Table 4.2 Total run-time for the computation of $PC + MB$ for the dataset with $n = 3,000$ genes and varying number of observations denoted by m.

No. of CPU cores	Time [s] $m = 250$	Time [s] $m = 500$
16	1,530	3,105
32	781	1,594
64	396	805
128	205	413
256	104	209
512	54	108

4.2.4.3 Performance Analysis for Varying Number of Observations

In the second experiment, we examined how the number of observations affects the scalability of our algorithms. The run-time as a function of the number of processor cores used are shown in Table 4.2. The same data is plotted in Figure 4.2 to show relative speedup when compared to a 16-core run. Similarly, these results were generated using the standard significance level cutoff of 0.001.

We evaluated the performance of our algorithms using two datasets with a fixed number of genes $n = 3,000$ and a varying number of observations $m = 250$ and $m = 500$, respectively. In order to compute each association measure (in our case equivalent to computing the G^2-statistic), the n-by-m input data-matrix is scanned. Thus, a smaller number of observations yields lower computational complexity, while leaving the communication complexity unaffected. This could potentially limit scalability. The results in Table 4.2 indicate that for a fixed number of variables, the run-time increases linearly as the number of observations increases, and as can be seen from Figure 4.2, the relative speedup remains nearly linear on up to 512 cores for both datasets.

Table 4.3 Run-times for the parallel inference of PCs and MBs on a genome scale *Arabidopsis thaliana* data set.

Phase	Time [s]
$\overline{MMPC}$ Phase 1	6,838
$\overline{MMPC}$ Phase 2	0.45
Rest of $MMPC$	2,505
Total time for PCs	9,344
Total time for MBs	3.71
Total time for PCs + MBs	9,347

4.2.4.4 Whole-genome BN Structure Learning

We demonstrate the applicability of our methods to a real-world problem by learning the whole-genome skeleton network of the model plant *Arabidopsis thaliana*. Using the *whole-genome* dataset, the full set of PCs and MBs were inferred on 255 nodes totaling $4,080$ cores. The results are summarized in Table 4.3. As expected, the most time-consuming part corresponded to phase 1 in the $\overline{MMPC}$ algorithm. The second phase took negligible time indicating there were few false positives that were removed during this stage. It also reflects the power of storing and reusing the *MinAssoc* computations. Computing MBs took negligible additional time, once the PC sets were computed. Our implementation allowed for the inference of the whole-genome skeleton BN in approximately 2 hours and 40 minutes.

4.3 Phase 2: Parallel Structure Learning for Large Scale Bayesian Networks

4.3.0.5 Contributions

In this section, we present a parallel approach to build large-scale BNs that combines a heuristic framework for scaling with the precision of exact learning. In this approach, we first

identify the set of candidate parents for each node in the network with the goal that the set so identified is a superset of the true parents (for subsequent correctness) and is as small as possible (for computational efficiency). Taking these sets as basis, we develop a parallel exact algorithm for collectively inferring the true parents of each node in the network. We expect that the resulting network should be of higher quality due to the utilization of the exact algorithm in refining the candidate parents sets. Nevertheless, no algorithm can guarantee optimality of the network. Sources of error in this approach include non-inclusion of actual parents in the candidate parent sets and how faithfully the scoring function models the interactions. As a result, the network derived by inferring the parent sets may contain cycles. We augment this with a parallel algorithm to detect and break cycles to result in a DAG that is returned as the inferred structure. We apply the presented approach to reconstruct gene networks of the model plant organism *Arabidopsis thaliana*.

4.3.1 Problem Statement and the Proposed Approach

Given a set $\mathcal{X}$ of n variables and a fully-observed data matrix $D_{n\times m}$ we wish to find the DAG N such that it optimizes a scoring function $Score(N)$ 1.2.2. Our work draws upon the two stage algorithm presented by Peng *et al.* [Peng and Ding, 2003], which is reported to produce results comparable to or better than other large-scale BN learning tools. However, it is not a direct parallelization of this work and employs a very different strategy for the first stage in order to inject an optimal exact structure learning algorithm for improving quality, while developing a parallelization of the second stage. We briefly review Peng *et al.*'s sequential algorithm next.

4.3.1.1 Overview of the Sequential Approach

In this approach, structure learning is carried out by independently finding the locally optimal parents set for each node in the network. The network is inferred by combining the results from this stage for all nodes in the network. The resulting network is not necessarily a DAG and may contain cycles. In the second stage, cycles are broken until the graph becomes acyclic while the score of the overall network is maximized.

Finding Locally Optimal Parents To pre-compute a set of locally-optimal parents for each variable in the network, a simple incremental routine is employed. The parents set of each node is initialized to the empty set. In each iteration, the next single best parent for a given node is added to the current parents set. The search continues until the overall score of the variable given its parents $s(X, Pa(X))$ can no longer be improved.

Minimal Score-loss Cycle Elimination Precomputing locally optimal parents as described above may result in a network containing cycles. To eliminate a simple cycle, it is sufficient to remove any of the edges in the cycle. The removal of any given edge (X_i, X_j) is equivalent to removing node X_i from the parents set of X_j. The score-loss function δ_{ij} provides an estimate of the reduction in score, which may result from this change:

$$\delta_{ij} = s(X_j, Pa(X_j)) - s(X_j, Pa(X_j) \backslash \{X_i\}).$$

In simple cycles, the edge with smallest score loss is removed. In nested cycles, removing an edge shared by multiple cycles could break more than one cycle. Thus, in the presence of nested cycles the edge shared by most cycles is first identified. Then the score loss of this edge is compared to the sum of smallest score loss edges in each of the participating cycles. If the score loss resulting from the removal of the most shared edge is minimal, then this edge

is removed and the remaining cycles are re-valuated in a similar manner. Otherwise, in each cycle the edge with the smallest score loss is removed.

Short-Cycle-First Heuristic: We next comment on how and in what order cycles of different lengths are detected. Cycles in a given graph can be identified by exponentiation of its adjacency matrix. If $M_{n\times n}$ is the adjacency matrix of graph G over n variables and the trace of $M^q_{n\times n}$ is not 0, then G contains cycles of length q. Peng *et al.* [Peng and Ding, 2003] make the observation that when cycles of different lengths are present, breaking short cycles first is more efficient than longer ones, as this results in a higher probability that more than one cycles are eliminated this way. Thus they identify cycles efficiently by exponentiation of the adjacency matrix from lowest to highest power.

After cycles are detected, they can be easily identified using depth first search. The multiplicity of each variable (the value in the main diagonal in $M^q_{n\times n}$ for each node) indicates the number of cycles in which the node participates and is used for the identification of the most shared edge in minimal score-loss cycle elimination.

4.3.2 Overview of the Parallel Approach

While we follow the overall framework of Peng *et al.* [Peng and Ding, 2003], we propose a different strategy for the first stage to more accurately determine the locally optimal parents of each node. This is carried out in two steps. Let n denote the number of variables/nodes in the BN. In the first step, we compute the candidate parent set of each node in the network. The goal here is to reduce the size of each candidate parent set from the maximum possible $n-1$ to as small a size as possible, without running into the risk of omitting a legitimate parent. This can be achieved using a variety of algorithms. For instance, Nikolova *et al.* [Nikolova and Aluru, 2011] present a parallel algorithm to compute candidate parent sets based on the

MMPC algorithm [Tsamardinos et al., 2006]. This fits well with the proposed approach as the entire framework will then run in parallel.

In other cases, there may be domain-specific strategies for identifying candidate parent sets. For example, the specific application of interest in our work is to infer gene regulatory networks. For this problem, it is natural to identify candidate parent sets based on pairwise correlations – if two genes are co-expressed as per a correlation measure, then each gene is put in the candidate parent set of the other. Correlations can be computed using multiple measures such as Pearson correlation, Spearman correlation, etc. In this work, we use the more rigorous parallel information-theoretic approach proposed by Zola *et al.* [Zola et al., 2010], that uses pairwise mutual information to detect nonlinear interactions, removes indirect interactions to further reduce candidate parent set sizes, and carries out permutation testing to ascertain the validity of predicted interactions.

The parallel network structure learning algorithm proposed in this work starts off with the candidate parents sets as input, obtained through one of the means explained above or another such method. The key contribution we make is to develop a parallel exact method to determine the locally optimal parents, and a parallel method to break cycles from the resulting network so obtained. We expect that the exact algorithm, which exhaustively evaluates all subsets of each candidate parents set, will yield a superior quality network.

In case of large networks, the sizes of some of the pre-computed candidate parents sets may be too large for the exact algorithm. In case of gene regulatory networks, it is highly likely that a gene with large node degree is a regulatory element that influences many other genes (hence the parent), and it is very unlikely that the expression of a gene is influenced by a large number genes. Hence, we compute locally optimal parents sets for all nodes whose candidate parents

sets do not exceed a threshold value, determined by the capability of the exact algorithm. For a node X_i whose candidate parents set size exceeds this threshold, we remove those nodes X_j which are already evaluated by the exact algorithm and the result affirms X_i is a parent of X_j. Thus, by eliminating children of X_i from its candidate parents set, we hope to reduce the size of this set to below the threshold required for the exact method. This procedure works because children of a node are often included in the candidate parents set of a node, due to the inability to distinguish between parents and children at this stage of the algorithm. For example, a correlation measure used to measure co-expression will infer an indirected network that is unable to differentiate between the parents and children of a node.

For the second stage of cycle removal, we propose a parallel method to carry out the cycle identification procedure by matrix exponentiation in short-cycle-first manner. Identified cycles are then resolved using the minimal score-loss criterion. The subsequent sections describe in detail our parallel algorithm.

4.3.3 Parallel Discovery of Locally Optimal Parents

We first present a parallel algorithm to compute the set of locally optimal parents for a single variable. Next, we demonstrate that it is advantageous to compute sets of locally optimal parents simultaneously for multiple variables. Finally, we present a scheduling scheme to pool together multiple variables and compute their locally optimal parents sets simultaneously.

4.3.3.1 Optimal Parents Set for a Single Variable

Let $CP(X_i) \subset \mathcal{X}$ be the candidate parents set for the variable X_i, and $A \subseteq CP(X_i)$. Using $D_{n\times m}$, a scoring function $s(X_i, A)$ determines the score of choosing A to be the set of parents for X_i; or equivalently, $s(X_i, A)$ specifies how well A predicts X_i given $D_{n\times m}$. The computation

of function s chosen from the most common scoring functions (BDe, MDL) requires $O(n \cdot m)$ time.

Definition 4.2 *Let $CP(X_i)$ be the candidate parents set of X_i and $A \subseteq CP(X_i)$. Let $F(X_i, A)$ denote the highest possible score that can be obtained by choosing parents of X_i from A; i.e.,*

$$F(X_i, A) = \max_{B \subseteq A} s(X_i, B).$$

A set B which maximizes the above equation is an optimal parents set for X_i ($OP(X_i)$) from the candidate parents subset A. While $F(X_i, A)$ can be computed by directly evaluating all $2^{|A|}$ subsets of A, it is advantageous to compute it based on the following recursive formulation. For any non-empty set A,

$$F(X_i, A) = \max \begin{cases} s(X_i, A) \\ \\ \max_{X_j \in A} F(X_i, A \backslash \{X_j\}) \end{cases}.$$

This formulation allows for a dynamic programming approach, which can be visualized as operating on the lattice L formed by the partial order "set inclusion" on the power set of $CP(X_i)$. Let $|CP(X_i)| = d$. Then the lattice L is a directed graph (V_L, E_L), where $V_L = 2^d$ and $(B, A) \in E_L$ if $B \subset A$ and $|A| = |B| + 1$. The lattice is naturally partitioned into levels, where level $l \in [0, d]$ contains all subsets of size l (see Figure 2.2).

A node $A \subseteq CP(X_i)$ at level l has l incoming edges from nodes $A \backslash \{X_j\}$ for each $X_j \in A$, and $d - l$ outgoing edges to nodes $A \cup \{X_k\}$ for each $X_k \in CP(X_i) \backslash A$. Exactly one F score is computed at node A, i.e $F(X_i, A)$. The computed value is further used by the nodes incident to the outgoing edges. Each level in the lattice can be computed concurrently, with data flowing

from one level to the next.

[Nikolova et al., 2009] present an optimal parallel algorithm for the problem of exact induction of optimal BN structures from data. While the memory requirements in the problem of exact BN structure learning would be prohibitive at the scale of interest in this work, the mapping strategy can be adapted for the problem of learning optimal parents sets. This is due to the significant reduction in memory requirements by 1) pre-selecting sets of CPs for each variable, and 2) relaxing the acyclicity constraint.

4.3.3.2 Run-time and Space Complexity

In [Nikolova et al., 2009] a similar subset lattice is efficiently mapped onto $p = 2^k$ processors, where $k < n$. Here we omit a comprehensive review of the method in [Nikolova et al., 2009] and instead present the differences unique to mapping the computation of a single F score. As $|PC(X_i)| = d$ we have a total of 2^d subsets to consider. The serial complexity of computing a single F score is given by

$$\sum_{i=0}^{d} (i + im) \cdot \binom{d}{i} = O\left(mn \cdot 2^d\right).$$

In parallel, we have a total of $k + 2^{d-k}$ parallel steps. At each step, at least one processor computes at least one F function, which yields $O(mn)$ time. Thus, the parallel run-rime is given by $O\left(mn \cdot \left(k + 2^{d-k}\right)\right)$. Given that the serial run time is $O\left(mn \cdot 2^d\right)$, optimal parallel run-time is $O\left(mn \cdot 2^{d-k}\right)$, which is achieved by our approach when $k + \log k < d$.

The total number of F-scores computed by the sequential algorithm is

$$\sum_{i=0}^{d} \binom{d}{i}.$$

Since storing only two levels at a time is sufficient, maximum number of F-scores is stored when $i = \lceil \frac{d}{2} \rceil$, which using Stirling's formula approximates to $\sqrt{\frac{2}{\pi}} \cdot \frac{1}{\sqrt{d}} \cdot 2^d$. In the parallel case, the total number of F-scores to be computed is

$$\sum_{i=0}^{d-k} \binom{d-k}{i}.$$

As before, the maximum storage needed at the same time is given by the summation of two consecutive terms in the above summation. Such terms are maximized when $i = \lceil \frac{d-k}{2} \rceil$. Then the maximum number of elements stored per processor becomes

$$\binom{d-k}{\lceil \frac{d-k}{2} \rceil},$$

or using Stirling's formula approximately $\sqrt{\frac{2}{\pi}} \cdot \frac{1}{\sqrt{d-k}} \cdot 2^{d-k}$. Thus, the ratio serial/parallel of number of stored elements yields

$$\frac{\sqrt{d-k}}{\sqrt{d}} \cdot 2^k,$$

which is optimal.

4.3.3.3 Simultaneous Computation of Optimal Parents Sets for Multiple Variables

Mapping

We next note that multiple variables may have candidate parents sets of the same size. For example, Figure 4.3 depicts the node degree distribution of candidate parents sets for two of the data sets of: $n = 250$ and $m = 200$, and $n = 500$ and $m = 100$. During the computation of any two OP sets, the communication, computation and local memory access

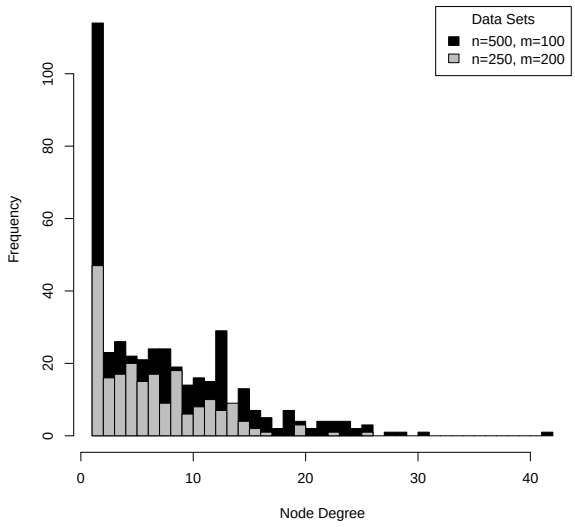


Figure 4.3 Node degree distribution for CPs for datasets of sizes $n = 250$, $m = 200$, and $n = 500$, $m = 100$.

patterns remain the same provided that the two candidate parents sets are of the same size. Thus, the computation of OP sets can be carried out simultaneously for each group (bucket) of CP sets of equal size. Therefore, all CP sets are first grouped by size. Then, all sets of size d from a given bucket are mapped to a d-dimensional lattice as described in Section V-A, using any 1-to-1 mapping function.

Scheduling In order to maintain optimal efficiency while computing multiple F scores simultaneously, the following scheduling can be utilized. Given $p = 2^k$ processors, from the run-time analysis it follows that any number of F scores for candidate sets of size d can be computed efficiently as long as $k + \log k < d$. Thus, after grouping all CP sets into buckets by size, if the optimality condition is met, we proceed to compute this bucket of F scores on all p processors. If the condition is not met, we select k' such that $k' + \log k' < d$ and then divide $p = 2^k$ into $2^{k-k'}$ groups of processors and proceed to compute the bucket of F scores optimally on the smaller sub-groups of processors. Since all subsets for which F scores are being computed

are of equal size, we have perfect balancing within each group of $2^{k'}$ processors, as well as between the $2^{k-k'}$ different subgroups of processors. We note that both p and any sub-group of processors $2^{k'}$ are powers of 2 and thus it is guaranteed that all processors can be divided between compute tasks.

Run-time and Space Complexity Consider computing a total of b F scores simultaneously, each for a CP of size d. Then, both the serial and parallel run-times are increased by a factor of b, resulting in $O\left(b \cdot mn \cdot 2^d\right)$ serially and $O\left(b \cdot mn \cdot \left(k + 2^{d-k}\right)\right)$ in parallel. The the parallel algorithm retains it optimality. The key reason for pooling F computations for nodes with CPs of same size is to save on communication costs. By pooling together computations for b nodes, each communication size will increase by a factor of b, while the number of parallel communications remains the same.

Similarly, both the serial and parallel space complexity increase by a factor of b, and are given by $b \cdot \sqrt{\frac{2}{\pi}} \cdot \frac{1}{\sqrt{d}} \cdot 2^d$ and $b \cdot \sqrt{\frac{2}{\pi}} \cdot \frac{1}{\sqrt{d-k}} \cdot 2^{d-k}$, respectively. This results in the same serial/parallel memory consumption ratio as shown for a single variable.

4.3.4 Resolving Highly Connected Nodes

In large networks it could be the case that some nodes would have large CP sets. For example, gene networks are usually scale-free, and thus the node degree distribution follows the power law: $P(d) = Ad^{-\gamma}$, where $P(d)$ denotes the fraction of nodes with degree d and $A \in [2, 3]$ is a constant [Barabasi and Oltvai, 2004]. For such cases, we propose a parallel technique for efficient node degree reduction.

Let d_{lim} be the maximum node degree allowed. First, OP sets are computed for all variables $X_k \in \mathcal{X}$ such that $CP(X_k) < d_{lim}$. Next, node degree reduction for all remaining variables $X_i \in \mathcal{X}$ such that $CP(X_i) \geq d_{lim}$ is attempted. For each such variable X_i, all candidate

parents $X_j \in CP(X_i)$ are considered. If $OP(X_j)$ has already been computed, and X_i has been selected as one of the optimal parents $X_i \in OP(X_j)$, then we deem X_j a child of X_i and remove it from the candidate parents set $CP(X_i)$, thus reducing $|CP(X_i)|$ by 1.

4.3.4.1 Parallel Connectivity Reduction

Consider revising the candidate parents set $CP(X_i)$ of variable X_i: $X_j \in CP(X_i)$ is removed from $CP(X_i)$ if $X_i \in OP(X_j)$. Serially, this check is performed for each CP set in each remaining bucket, for each variable within that set. We note that since each processor computes optimal parents sets for different variables, these sets are now distributed across all nodes. A trivial parallelization would require tracking where the optimal parents set resides, for each variable in every given CP set.

We perform this reduction for all variables and optimal parents sets simultaneously as follows. First, we transform all OP sets computed thus far and the remaining CP sets into pairs representation: $\forall X_k \in OP(X_j)$ create a pair (X_k, X_j), where the first variable indicates the optimal parent X_k, and the child X_j is recorded as the second variable. Such pairs are marked as generated from an OP set. Further, $\forall X_j \in PC(X_i)$ we create a pair (X_i, X_j), where the candidate parent X_j is now the second variable, and the candidate child X_i is the first. Such pairs are marked as generated from a CP set. The CP sets are assigned to processors in a round-robin fashion. Every processor generates OP pairs from its local OP sets, and CP pairs from the CP sets assigned to it. Once all pairs are generated, a global parallel sort is invoked, using the first variable as primary key, and the second variable as secondary key.

After sorting, all duplicates will appear consecutively. Duplicate pairs (X_i, X_j), such that each was generated by either OP or CP set, would appear only if the candidate child variable X_i was selected as optimal parent $X_i \in OP(X_j)$ for X_j. Thus, we remove all CP pairs

for which a duplicate OP pair was available and update the remaining CP lists. The pair-representation allows for the generation of OP and CP pairs by all processors simultaneously. Next, new buckets are formed from the remaining CP sets and OP sets are computed as described in 4.3.3.1.

4.3.4.2 Run-time and Space Complexity

Processors are assigned CP sets for CP-pair generation in a round-robin manner, with the aim of evenly distributing the number of resulting CP-pairs per processor. The total number of CP pairs is $\sum_{d=d_{lim}}^{d_{max}} b_d \cdot d$ where d_{max} is the highest node degree and b_d is the number of CP sets in a bucket with degree d. This number depends on the application and particular dataset. However, as our application shows, the total number of nodes with large connectivity is moderate (see Table 4.4). The OP sets reside on the processor with the highest rank within its local group of $2^{k'}$ processors. For CP sets of size $d \geq k$ where $k = \log p$, the resulting OP's will reside on the processor with the highest rank $p-1$. The number of OP's is clearly bounded by the total number of variables n. The size of OP is bounded by d_{lim}. Parallel sort is performed on all OP and CP pairs. In our application we use parallel sample sort.

4.3.5 Parallel Cycle Removal

The OP sets for all variables form a directed graph, which may have cycles. One way to globally detect all cycles in a directed graph is by exponentiating its adjacency matrix. Let $M_{n\times n} = \{m_{ij}\}$ be the adjacency matrix of the directed graph which corresponds to all optimal parents sets. If for some power $2 \leq q \leq n-1$ the trace of the matrix $M^q_{n\times n} \neq 0$, then all non-zero elements in the main diagonal $m_{ii} \neq 0$ of $M_{n\times n}$ participate in a cycle of length q. The multiplicity of each node (the value of m_{ii}) corresponds to the number of paths which go

through this node. Thus, exponentiating the adjacency matrix $M_{n\times n}$ to power q is sufficient for detecting 1) if cycles of length q are present, 2) which nodes participate in cycles of length q, and 3) how many cycles of length q pass through each of the participating nodes. Further, a depth first search (DFS) can be used to enumerate all cycles of length q. When cycles are detected, edges are removed following the minimal score-loss heuristic. This results in the final acyclic network.

We exponentiate $M_{n\times n}$ in parallel, in a short-cycle-first manner, starting from $q = 2$. By eliminating cycles in increasing cycle length order, we ensure that after breaking all cycles of length q, no cycles of length leq are present in $M_{n\times n}$. This guarantees, that we will need to exponentiate $M_{n\times n}$ at most $n - 2$ times.

4.3.6 Repair of Optimal Parents Sets

Once all cycles are broken, we revisit the variables which parents sets have been modified during the cycle breaking procedure. For each such variable, we re-optimize the F score within the modified candidate parents set.

4.3.7 Experimental Results

The presented algorithms were implemented in C++ and MPI. To assess the performance and applicability of the algorithms experiments were performed on a Cray system with two 12-core AMD 'ManyCores' 2.1-GHz processors and 32 GB DDR3 1333-MHz memory per node. Each MPI process was run with 8 GB of memory. To evaluate scalability we performed experiments on a medium-size data set of $n = 500$ variables and $m = 100$ observations.

Table 4.4 Candidate parents sets statistics for the test data set of size $n = 500$, $m = 100$.

Data set	$D_{500\times100}$
Num. Edges	1,643
Max Node Degree	42
Avg Node Degree	7.9
Num. CP Sets of size ≥ 1	415
Num. CP Sets of size ≥ 25	9

4.3.7.1 Data Preparation

A total of $3,546$ non-redundant Affymetrix expression profiles for *Arabidopsis thaliana* were collected from the NASCArray [NAS, 2009], AtGenExpress [TAI, 2009], ArrayExpress [Arr, 2009], and GEO [GEO, 2009] repositories and pre-processed as described in [Zola et al., 2010]. This resulted in a data-matrix of $3,137$ observations and $15,596$ probe-sets, mapping to $15,495$ genes. We created a medium-scale data set consisting of 500 genes and 100 observations by selecting subsets of genes and subsamples.

The values for each variable are the gene expression levels detected by microarray experiments, which are continuous. We represent gene expression using a multinomial model. After the normalization performed in [Zola et al., 2010] and reviewed in 2.5.2, we discretized the data into three levels, using the average expression value of each gene across experiments as control in a similar manner as in [Friedman et al., 2000]. The candidate parents were pre-computed using the parallel information-theoretic approach proposed by Zola *et al.* [Zola et al., 2010] with statistics summarized in Table 4.4.

Table 4.5 Total run-time in seconds for the computation of optimal parents sets of size no more than 25 (OP1), node degree reduction (DR), computation of remaining optimal parents sets (OP2), and cycle removal (CYC), for the data set with $n = 500$ genes and $m = 100$ observations.

No. Cores	OP1	DR	OP2	CYC
16	2717	0.011	2904	25.09
32	1365	0.005	1457	16.40
64	705	0.006	740	11.00
128	377	0.010	390	11.63
256	173	0.014	184	7.02
512	87	0.030	91	6.85
1024	49	0.072	50	7.42

4.3.7.2 Scalability Analysis

In this experiment we evaluate the performance of the presented algorithms as a function of the number of processor cores for a data set with $n = 500$ genes and $m = 100$ observations. The run-times for the different stages of the parallel framework are shown in Table 4.5. The same data for computing optimal parents sets before and after degree reduction is plotted in Figure 4.4 to show relative speedup when compared to a 16-core run. These results were generated using $d_{lim} = 25$.

The results show that computing optimal parents for this test data is the most time consuming task. As shown in Table 4.4, of the total 415 variables with CP sets of size 1 or larger, only 9 are of size greater or equal to 25. Nevertheless, due to the exponential complexity of computing optimal parents, the time required to compute the remaining 9 large sets is on average longer than the 406 smaller sets. In this test case, the time for degree reduction is negligibly small. This is to be expected as CP pairs are created for only 9 variables, and the resulting network is sparse. However, as node degree reduction is dominated by parallel sort,

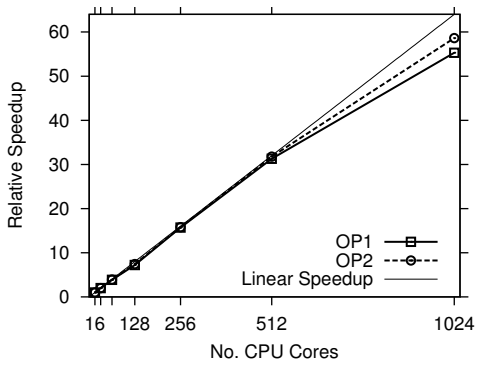


Figure 4.4 Relative speedup when compared to a 16-core run for the data set with $n = 500$ genes and $m = 100$ observations, for the computation of optimal parents sets from CP sets of size no more than 25 (OP1) and the remaining ones (OP2).

we expect it to have good strong scaling for denser networks. Similarly, the adjacency matrix exponentiation and cycle removal is relatively fast. This would change as a function of the sparsity of the network. Finally, we note that the computation of optimal parents exhibits nearly perfect scaling on up to 1024 cores.

4.3.7.3 Quality Assessment of Learned Networks

To assess the quality of the learned networks we compare the performance of our algorithm to the exact BN structure learning algorithm using the BDe scoring function. To make this comparison possible, we used a small synthetically generated dataset, where we controlled the number of variables and observations, and which was possible to compute exactly. We used $SynTReN$ to generate a network of $n = 24$ variables and $m = 1000$ observations (same network as shown in Figure 2.6 in Chapter 2). The heuristic algorithm produced network with total score within 0.94% of optimal.

4.4 Conclusions and Future Work

Large-scale Bayesian network inference is an important problem of relevance to many disciplines. In this chapter, we presented a two-phase parallel approach for computing (1) the skeleton of a Bayesian network, that is the parents and children set of each node and for computing the Markov boundary of each node, and (2) a parallel BN structure learning algorithm that combines the heuristic approach of resolving cycles in the global network with an exact algorithm that refines pre-computed candidate parents sets to produce the final BN network. To the extent that the pre-computed sets of parents contain the true parents, the exact algorithm on the constrained sets should produce results identical to those obtained as if the exact algorithm is run on the whole network. We demonstrate that the resulting algorithm is scalable and fast. We show that these computations scale well and permit inference of large Bayesian networks using parallel computers.

CHAPTER 5. Summary and Future Directions

In this work, we presented a set of parallel algorithms for the Bayesian network (BN) structure learning problem, and their application to inferring transcriptional networks in *Arabidopsis thaliana*.

We first described a parallel algorithm for exact structure learning. This algorithm is an efficient parallelization of a state-of-the-art dynamic programming approach, which explored the combination of a canonical representation of directed acyclic graphs (DAGs), and the decomposability of the scoring function. We used the canonical representation of the network to devise a mapping of the computations to p processors, which is work optimal for $2^n > p \log p$. We decomposed the problem into $2^{n-\log_2 p}$ subproblems, and devised an order in which these can be efficiently pipelined. Our algorithm is communication efficient and uses space within 1.41 of the optimal. Our experimental results demonstrate that it achieves good scaling for up to $2,048$ processors. It allowed the induction of a BN over 33 variables in only 1 hour and 14 minutes using the MDL scoring function.

Next, we extended this work to the case of bounded node in-degree, where a limit d on the number of parents per variable is imposed. We characterized the algorithm's run-time behavior as a function of d, establishing the range $\left[\frac{1}{3}n - \log mn, \lceil \frac{n}{2} \rceil\right)$ of values for d where it affects asymptotic run-time performance. Consequently, two plateau regions are identified for $d < \frac{1}{3}n - \log mn$ and $d \geq \lceil \frac{n}{2} \rceil$, where the run-time complexity remains the same as respectively

for $d = 1$ and $d = n - 1$.

Further, we evaluated the biological relevance of learning transcriptional networks from large mixtures of microarray experiments. We applied our software to data from a targeted set of experiments, as well as to a collection of all publicly available microarray experiments for 13 genes known to be involved in the Carotenoid Biosynthesis pathway of *Arabidopsis thaliana*. We compared our findings to another method based on mutual information. Our evaluation based on prior knowledge of the metabolic pathway of Carotenoid Biosynthesis suggests that we are able to generate relevant networks.

Finally, we presented a two-phase heuristic approach to BN structure learning. In the first phase, a set of candidate parents for each variable in the domain is precomputed using a state-of-the-art heuristic. The key idea in this work is to transform the problem into a parallel sorting problem, which is done efficiently on parallel computers. We were able to generate the candidate parents for the whole-genome dataset ($15,596$ genes and $3,137$ observations) of *Arabidopsis thaliana* in approximately 2 hours and 40 minutes.

In the second phase, we combine heuristic framework for scaling with precision of exact learning. With the assumption that the pre-computed set of parents contains the true parents, we learned the optimal parents set for each variable exactly. We then proceeded to break any existing cycles in parallel while minimizing score loss, thus maximizing the overall network score. Our algorithm achieved nearly perfect scaling on up to $1,024$ cores and was capable of learning networks for as many as 500 variables. When compared to the performance of an exact BN learning algorithm on a small dataset, our method learned a network with a score within 0.94% of the optimal score.

All of the devised algorithms were implemented in MPI and C++ and tested on various

platforms (BlueGene/L, BlueGene/P, Sun and Cray AMD clusters).

Even though multiple methods have been developed in the last decade for gene network reconstruction from gene expression data, the field is still in its infancy. Computational predictions made on biological data are difficult to interpret and validate. In this light, we outline several future directions for improving the quality and biological relevance of the learned networks, as well as scalability to large data.

First and foremost, while BNs have the capability of learning biologically meaningful transcriptional networks, gene expression data only gives a partial picture of the processes in the cell. In order to begin to understand the regulatory effects in complex systems, different types of data that capture epigenetic effects, such as methylation, need to be combined with transcriptional data. Bayesian networks allow for integration of multiple types of data by admitting a mixture of categorical and numerical variables. An exciting future direction is to apply the BN model to such cross-data type sets.

Additional modeling challenges for BNs are introduced by the presence of self-loops. Dynamic Bayesian networks (DBNs) could overcome these challenges, as they have the ability to model loops. However, DBNs require time-series data, which, if collected from multiple experiments, could be very difficult to pre-process and use. Scalability to large data may also pose a challenge.

In the case of exact learning, our experiments show that memory is the bottleneck currently limiting the scalability of our parallel algorithm. While branch-and-bound approaches may be able to decrease the number of computations and potential memory requirement, load balancing and parallel efficiency are likely to pose a serious challenge. Improvements made to address these issues have the potential to resolve the memory bottleneck and allow for large scale exact

learning.

In the case of heuristic learning, a thorough comparison of the different approaches for pre-computing candidate parents may lead to improvements in the quality of the inferred networks. Further biological evaluation of the generated large-scale networks is also needed to improve upon the ability of existing models to generate biologically relevant networks.

Bibliography

Plant Metabolic Network (PMN). URL http://pmn.plantcyc.org/PLANT/new-image?type=PATHWAY&object=CAROTENOID-PWY&detail-level=2&ENZORG=ORG-5993.

EMBL-EBI ArrayExpress. http://www.ebi.ac.uk/microarrayas/aer/, 2009.

NCBI Gene Expression Omnibus. http://www.ncbi.nlm.nih.gov/geo/, 2009.

NASC European Arabidopsis Stock Centre. http://www.arabidopsis.info/, 2009.

TAIR. http://www.arabidopsis.org/, 2009.

H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, December 1974. ISSN 0018-9286. doi: 10.1109/TAC.1974.1100705. URL http://dx.doi.org/10.1109/TAC.1974.1100705.

M. Aluru, J. Zola, D. Nettleton, and S. Aluru. Reverse engineering and analysis of the arabidopsis whole-genome network. *Nucleic Acids Research*, Under Review.

A.-L. Barabasi and Z. N. Oltvai. Network biology: understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2):101–113, February 2004. ISSN 1471-0056. doi: 10.1038/nrg1272. URL http://dx.doi.org/10.1038/nrg1272.

S. Basilio, I. Inaki, and P. Larranaga. Medical Bayes networks. In *ISMDA '00: Proceedings of the First International Symposium on Medical Data Analysis*, pages 4–14, 2000.

K. Basso, A. A. Margolin, G. Stolovitzky, U. Klein, R. Dalla-Favera, and A. Califano. Reverse engineering of regulatory networks in human b cells. *Nature Genetics*, 37(4):382–390, March 2005. ISSN 1061-4036. doi: 10.1038/ng1532. URL http://dx.doi.org/10.1038/ng1532.

R.R. Bouckaert. *Bayesian Belief Networks: from Construction to Inference*. PhD thesis, Utrecht, Netherlands, 1995.

A. Brazma, H. Parkinson, U. Sarkans, M. Shojatalab, J. Vilo, N. Abeygunawardena, E. Holloway, M. Kapushesky, P. Kemmeren, G. G. Lara, A. Oezcimen, P. Rocca-Serra, and S. Sansone. Arrayexpress–a public repository for microarray gene expression data at the ebi. *Nucl. Acids Res.*, 31(1):68–71, January 2003. doi: 10.1093/nar/gkg091. URL http://dx.doi.org/10.1093/nar/gkg091.

Christopher I. Cazzonelli, Andrea C. Roberts, Melanie E. Carmody, and Barry J. Pogson. Transcriptional control of set domain group 8 and carotenoid isomerase during arabidopsis development. *Molecular Plant*, 3(1):174–191, 2010. doi: 10.1093/mp/ssp092. URL http://mplant.oxfordjournals.org/content/3/1/174.abstract.

X. Chen, G. Anantha, and X. Wang. An effective structure learning method for constructing gene networks. *Bioinformatics*, 22:1367–1374, 2006.

J. Cheng. Learning bayesian networks from data: An information-theory based approach. *Artificial Intelligence*, 137(1-2):43–90, May 2002. ISSN 00043702. doi: 10.1016/S0004-3702(02)00191-1. URL http://dx.doi.org/10.1016/S0004-3702(02)00191-1.

D. M. Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3:507–554, 2002.

D. M. Chickering, D. Heckerman, and D. Geiger. Learning Bayesian networks is NP-hard. Technical Report MSR-TR-94-17, Microsoft Research, 1994.

G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.

T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 2nd edition, 2006.

D. J. Craigon, N. James, J. Okyere, J. Higgins, J. Jotham, and S. May. Nascarrays: a repository for microarray data generated by nasc's transcriptomics service. *Nucl. Acids Res.*, 32(suppl_1):D575–577, January 2004. doi: 10.1093/nar/gkh133. URL http://dx.doi.org/10.1093/nar/gkh133.

G. Elidan and N. Friedman. Learning hidden variable networks: The information bottleneck approach. *J. Mach. Learn. Res.*, 6:81–127, 2005. ISSN 1532-4435.

N. Friedman. Inferring cellular networks using probabilistic graphical models. *Science*, 303: 799–805, 2004.

N. Friedman and D. Koller. Being Bayesian about network structure. a Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50(1-2):95–125, 2003.

N. Friedman, I. Nachman, and D. Pe'er. Learning Bayesian network structure from massive datasets: The sparse candidate algorithm. pages 206–215.

N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using Bayesian networks to analyze expression data. *Journal of Computational Biology*, 7(3):601–620, August 2000. URL http://citeseer.ist.psu.edu/friedman99using.html.

A. Fuente, P. Brazhnik, and P. Mendes. Linking the genes: inferring quantitative gene networks from microarray data. *Trends in genetics : TIG*, 18(8):395–398, August 2002. ISSN 0168-9525. URL http://view.ncbi.nlm.nih.gov/pubmed/12142007.

M. Grzegorczyk and D. Husmeier. Improving the structure MCMC sampler for Bayesian networks by introducing a new edge reversal move. *Machine Learning*, 71(2-3):265–305, 2008.

Alexander Hartemink. Banjo: Bayesian Network Inference with Java Objects. URL http://www.cs.duke.edu/~{}amink/software/banjo.

D. Heckerman, D. Geiger, and D. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995a.

D. Heckerman, A. Mamdani, and M. P. Wellman. Real-world applications of Bayesian networks. *Commun. ACM*, 38(3):24–26, 1995b.

M. Koivisto. Advances in exact Bayesian structure discovery in Bayesian networks. In *Proceedings of the 22nd Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)*, 2006.

M. Koivisto, K. Sood, and M. Chickering. Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, 5:2004, 2004.

Spyros Kotoulas and Ronny Siebes. The chipping forecast I. *Nature genetics*, 21(suppl):1–60, November 1999.

W. Lam and F. Bacchus. Learning Bayesian belief networks an approach based on the MDL principle. *Computational Intelligence*, 10:269–293, 1994.

D. Marbach, R. J. Prill, T. Schaffter, C. Mattiussi, D. Floreano, and G. Stolovitzky. Revealing strengths and weaknesses of methods for gene network inference. *Proceedings of the National Academy of Sciences*, 107(14):6286–6291, April 2010. ISSN 1091-6490. doi: 10.1073/pnas.0913357107. URL http://dx.doi.org/10.1073/pnas.0913357107.

A. Moore and W.-K. Wong. Optimal reinsertion: A new search operator for accelerated and more accurate Bayesian network structure learning. In T. Fawcett and N. Mishra, editors, *Proceedings of the 20th International Conference on Machine Learning (ICML '03)*, pages 552–559, Menlo Park, California, August 2003. AAAI Press.

R. E. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2003.

O. Nikolova and S. Aluru. Parallel discovery of direct causal relations and markov boundaries with applications to gene networks. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 512 –521, sept. 2011. doi: 10.1109/ICPP.2011.49.

O. Nikolova, J. Zola, and S. Aluru. A parallel algorithm for exact Bayesian network inference. In *HiPC*, pages 342–349, 2009.

S. Ott, S. Imoto, and S. Miyano. Finding optimal models for small gene networks. In *Pacific Symposium on Biocomputing*, pages 557–567.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

D. Pe'er, A. Regev, G. Elidan, and N. Friedman. Inferring subnetworks from perturbed expression profiles. *Bioinformatics*, 17(suppl 1):S215–S224, June 2001. ISSN 1367-4803. doi: 10.1093/bioinformatics/17.suppl_1.S215. URL http://dx.doi.org/10.1093/bioinformatics/17.suppl_1.S215.

D. Pe'er, A. Tanay, and A. Regev. Minreg: A scalable algorithm for learning parsimonious regulatory networks in yeast and mammals. *Journal of Machine Learning Research*, 7:167–189, 2006.

J. M. Pena, R. Nilsson, J. Bjorkegren, and J. Tegner. Towards scalable and data efficient learning of Markov boundaries. *International Journal of Approximate Reasoning*, 45(2):211 – 232, 2007. ISSN 0888-613X. doi: DOI:10.1016/j.ijar.2006.06.008. URL http://www.sciencedirect.com/science/article/B6V07-4KM9W4W-2/2/1b1c576e4d153acd0e4bc56370acbab5. Eighth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2005).

H. Peng and C. Ding. Structure search and stability enhancement of Bayesian networks. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 621 – 624, nov. 2003. doi: 10.1109/ICDM.2003.1250992.

J.J. Rice, Y. Tu, and G. Stolovitzky. Reconstructing biological networks using conditional correlation analysis. *Bioinformatics*, 21:765–73, 2005.

R. W. Robinson. Counting labeled acyclic digraphs. In F. Harary, editor, *New Directions in Graph Theory*. New York: Academic Press, 1973.

M. Schena, D. Shalon, R. W. Davis, and P. O. Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 270(5235):467–470, October 1995. URL http://www.sciencemag.org/cgi/content/full/270/5235/467.

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. doi: 10.2307/2958889. URL http://dx.doi.org/10.2307/2958889.

E. Segal, Michael S., A. Regev, D. Pe'er, D. Botstein, D. Koller, and N Friedman. Module networks: identifying regulatory modules and their condition-specific regulators from gene expression data. *Nature genetics*, 34(2):166–176, June 2003. ISSN 1061-4036. doi: 10.1038/ng1165. URL http://dx.doi.org/10.1038/ng1165.

T. Silander and P. Myllymaki. A simple approach for finding the globally optimal Bayesian network structure. In *Proceedings of the 22nd Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)*, 2006.

P. Spirtes and C. Meek. Learning bayesian networks with discrete variables from data. In *1st International Conference on Knowledge Discovery and Data Mining (KDD-95)*, Montreal, Quebec, 1995.

P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search, 2nd Edition*, volume 1 of *MIT Press Books*. The MIT Press, June 2001.

Y. Tamada, S. Imoto, H. Araki, M. Nagasaki, C. Print, D. S. Charnock-Jones, and S. Miyano. Estimating genome-wide gene networks using nonparametric Bayesian network models on massively parallel computers. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 8(3):683–697, May 2011. ISSN 1545-5963. doi: 10.1109/TCBB.2010.68. URL http://dx.doi.org/10.1109/TCBB.2010.68.

M. Teyssier and D. Koller. Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *In UAI*, pages 584–590, 2005.

I. Tsamardinos, L. Brown, and C. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65:31–78, 2006. ISSN 0885-6125. URL http://dx.doi.org/10.1007/s10994-006-6889-7. 10.1007/s10994-006-6889-7.

T. Van den Bulcke, K. Van Leemput, B. Naudts, et al. SynTReN: a generator of synthetic gene expression data for design and analysis of structure learning algorithms. *BMC Bioinformatics*, 7:43, 2006. URL http://dx.doi.org/10.1186/1471-2105-7-43.

T. Verma and J. Pearl. Causal networks: semantics and expressiveness. In *UAI*, pages 69–78, 1988.

T. Verma and J. Pearl. A theory of inferred causation. In *Second Internation Coference on the Principles of Knowledge Representation and Reasoning*, Cambridge, Massachusetts, April 1991. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.7221.

M. Wang, Z. Chen, and S. Cloutier. A hybrid Bayesian network learning method for constructing gene networks. *Comput Biol Chem*, August 2007. ISSN 1476-9271. doi: 10.1016/j.compbiolchem.2007.08.005. URL http://dx.doi.org/10.1016/j.compbiolchem.2007.08.005.

J. Zola, M. Aluru, A. Sarje, and S. Aluru. Parallel information-theory-based construction of genome-wide gene regulatory networks. *IEEE Transactions on Parallel and Distributed Systems*, 21:1721–1733, 2010. ISSN 1045-9219. doi: http://doi.ieeecomputersociety.org/10.1109/TPDS.2010.59.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Srinivas Aluru, for introducing me to the world of high performance scientific computing, for urging me to develop as an independent researcher, for encouraging my interest in difficult algorithmic questions, and for showing me how to work effectively towards meeting important deadlines. I also thank him for helping me develop my writing skills, as well as, for the full support of my research assistantship and multiple conference travels. I would especially like to thank him for his wise words during some of my hardest times at Iowa State University, and teaching me how to remain focused on research in a highly stressful environment.

I would like to thank Dr. Patrick Schnable for the enormous inspiration to pursue bioinformatics as early as my senior year of college, and later during my rotation in his group.

I thank Dr. Jaroslaw Zola for mentoring me during most of my time in Dr. Aluru's group. For showing me the "right way" to implement and test parallel codes, for numerous discussions of research ideas, and his rigorous feedback on presentations and manuscripts.

I would like to thank Drs. Chris Bishop, John Winn, Nevena Lazic, and John Guiver, for the amazing opportunity to work with them at Microsoft Research at Cambridge UK. For introducing me to the world of causality, approximate Bayesian inference methods, and for inspiring my interest in bio-medical applications.

I thank Dr. Karin Dorman, and my mentor and friend Dr. Elena Zheleva, for their advice

and perspective on research and life.

I would like to thank Dr. Maneesha Aluru for collaborating on the application of Bayesian structure learning methods to Arabidopsis thaliana. I would like to thank all the POS committee members for their participation in my preliminary exam and final defense, and for providing valuable suggestions. I would like to thank the BCB program for the partial support of my assistantship through the first year of my Ph.D. and via the James Cornette Fellowship, and BCB and ECprE department for funding some of my conference travels.

I would like to thank my colleagues Abhinav Sarje, Xiao Yang, Matt Regennitter, Ben Jackson, Indranil Roy, and Sriram Chockalingam, my grad-school sister Kyoungmin Roh, and my classmates Fadi Towfic and Xue Li, and Deepak Reyon, Divya Mistry, Rasna Walia. I warmly thank Trish Stauble for her amazing presence in the BCB program, for making little miracles happen in assisting graduate students.

Last but not least, I warmly thank Rachel Meyers, Matthew Mauk, Mark Looney, Valentina Salotti, Marcus Diem, Laura Smarandescu, Kerry Campbell, Janine Cheng, and Simge Cinar, for being wonderful friends and keeping my spirits up. I thank Scott Boyken for his faith in me, for his excitement about my work, and for his endless support and positiveness during the last years of my Ph.D.

Finally, I thank my parents Svetlana and Hristo, and my brother Nikolay, for being there for me all the way, for their unlimited support, inspiration, and wisdom.